National Imagery and Mapping Agency

NATIONAL IMAGERY TRANSMISSION FORMAT STANDARD (NITFS)

BANDWIDTH COMPRESSION STANDARDS AND GUIDELINES

Review Draft 1

13 October 1997

Working Draft

CHANGE LOG

Revision or Change Notice	Date	Pages Changed	Authority

EFFECTIVITY LOG

No.	SCN	Effective		Title
		Per	On	

Table of Contents

List of Figures

List of Tables

TBD/TBR LISTING

Paragraph	Page No	TBD	TBR	Description
Table 3-2	14		R001	MIL-STD-188-198 to ISO 10918-1 approval
Table 3-2	14		R002	JPEG Lossless to ISO 10918-1/4 approval
Table 3-2	14		R003	MIL-STD 188-196 to CCITT Rec T4 approval
Table 3-2	14		R004	MIL-STD-188-199 to ISO 12087-5 approval

1.0 INTRODUCTION

1.1 Purpose

This document defines the Bandwidth compression standards, conventions and guidelines required for use by the National Imagery Transmission Format Standard (NITFS). It includes specifications on those standards, and implementation related conventions and guidelines to improve interoperability of NITFS compressed files within the United States Imagery and Geospatial System (USIGS). Because the standards environment is constantly changing, this document will be updated on a periodic basis to adjust for refinements of existing standards and profiles of standards, as well as for the emergence of new standards and technologies. Thus, this document shall contain references to Military Standards, commercial standards from the International Standards Organization (ISO), and guidelines to support vendors and developers as they develop NITFS tools and applications.

1.2 Scope

This document focuses on those compression-related standards, and relevant guidelines pertaining to the NITFS compression and decompression requirements as defined in MIL-STD-2500B (NITF 2.1) and the NITFS Standards Compliance and Interoperability Certification Test & Evaluation Program Plan. It shall refer to Military Standards rather than duplicate them here in their entirety, but will include, in total, profiles of those commercial standards that have been adopted by the NITFS for implementation, and will be the authoritative reference for implementation and procurement activities.

1.3 Applicability

The standards, conventions and guidelines defined in this document apply to the planning, development, test, evaluation, and operation of imagery and geospatial systems that generate (“pack”) or receive (“unpack”) NITFS files within the USIGS environment. Specific systems implementation requirements, such as when one or more compression standards or profiles are to be implemented by a USIGS system, are not provided here, but are found in other USIGS documents, such as the USIGS Interoperability Profile (UIP),or the USIGS Technical Architecture (UTA) for example. Effectivities provided in this document relate specifically to the standards development, maturation, and approval process, not the systems implementation requirements specifically.

1.4 Objectives

As the NITFS suite migrates from the specification of stovepipe bandwidth compression implementations and Military Standards, the community requires a configuration managed and systematically maintained set of standards and guidelines supporting and managing this direction. The intent of this document is to provide users, program managers, developers, commercial vendors, and decision makers the following:

- the specific set of military compression standards and ISO compression standards that are being, or to be, implemented within the NITFS and within the USIGS community as required by MIL-STD-2500B

- guidelines that will assist in the implementation or interoperability of bandwidth compression as required by MIL-STD-2500B

- a reliable “road map” of the changes planned to the existing standards and their successors in conjunction with the NITFS Program Plan (NIMA document NNPP) and the “Emerging Standards” section of the USIGS Technical Architecture (UTA)

This document will provide a clear understanding of the changes planned to the standards, be they Military Standards or international standards profiles, as well as anticipated dates that the changes will be under configuration control. It is expected that revisions of this document will be provided to the USIGS community in intervals of six months to a year, so that users have enough time to assess impacts of proposed changes, as well as for program managers to budget the required changes into their program plans and budgeting cycles.

2.0 APPLICABLE DOCUMENTS

JIEO Plan 9000	Department of Defense and Intelligence Community Imagery Information Technology Standards Management Plan, 01 November 1995.
CJCSI 6212.01A	Compatibility, Interoperability, and Integration of Command, Control, Communications, Computers, and Intelligence Systems, 30 June 1995.
JIEO Circular 9002	Requirements Assessment and Interoperability Certification of C4I and AIS Equipment and Systems, 23 January 1995.
JIEO Circular 9008	NITFS Certification Test and Evaluation Program Plan, 30 June 1993, with Errata Sheet dated 24 July 1996.
DOD/JTA V1.0	Department of Defense Joint Technical Architecture Version 1.0, 22 August 1996

(Requests for copies of the above policy and planning documents may be addressed to the Joint Interoperability Test Command, NITFS Test Facility, Building 57305, Fort Huachuca, AZ 85613-7020).

2.1 Military Standards and Handbooks

MIL-STD-2500A	National Imagery Transmission Format (Version 2.0) for the National Imagery Transmission Format Standard, 12 October 1994 with Notice 1, 7 February 1997 and Notice 2, 26 September 1997.
MIL-STD-2500B	National Imagery Transmission Format (Version 2.1) for the National Imagery Transmission Format Standard, 22 August 1997.
MIL-STD-188-196	Bi-Level Image Compression for the National Imagery Transmission Format Standard, 18 June 1993 with Notice 1, 27 June 1996.
MIL-STD-188-198A	Joint Photographic Experts Group (JPEG) Image Compression for the National Imagery Transmission Format Standard, 15 December 1993 with Notice 1, 12 October 1994 and Notice 2, 14 March 1997.

MIL-STD-188-199	Vector Quantization Decompression for the National Imagery Transmission Format Standard, 27 June 1994 with Notice 1, 27 June 1996.
MIL-STD-2301	Computer Graphics Metafile (CGM) Implementation Standard for the National Imagery Transmission Format Standard, 18 June 1993 with Notice 1, 12 October 1994.
MIL-STD-2045-44500	Tactical Communications Protocol 2 (TACO2) for the National Imagery Transmission Format Standard, 18 June 1993 with Notice 1, 29 July 1994 and Notice 2, 27 June 1996.
MIL-STD 188-197A	Adaptive Recursive Interpolated Differential Pulse Code Modulation (ARIDPCM) Compression Algorithm for the National Imagery Transmission Format Standard, 12 October 1994.
MIL-HDBK-1300A	Military Handbook for the National Imagery Transmission Format Standard (NITFS), 12 October 1994.

(Copies of the above military standards and handbooks are available from the Standardization Document Order Desk, 700 Robbins Avenue, Building 4D, Philadelphia, PA 19111-5094).

2.2 NIMA Specifications and Publications

NUAF	NIMA United States Imagery and Geospatial System (USIGS) Architecture Framework (NUAF), Draft.
N0101-A	Geospatial Image Access Services Specification (GIAS), 22 April 1997.
N0102-A	United States Imagery and Geospatial System (USIGS) Interoperability Profile (UIP), 22 July 1997.
NUTA	NIMA USIGS Technical Architecture (NUTA), 28 October 1997.
NPP DRAFT	NITFS Program Plan, 13 October 1997, Draft.

N0105 DRAFT	NITFS Standards Compliance and Interoperability Test and Evaluation Program Plan, Review Draft 5, 24 October 1997.
NPIAE	NIMA Profile for Imagery Archive Extensions (NPIAE) for the National Imagery Transmission Format Standard (NITFS), 26 September 1997.
NSDE/97	NIMA Support Data Extensions (SDE) (Version 1.2) for the National Imagery Transmission Format Standard (NITFS), 13 March 1997.
RASG-9606-001	Airborne Synthetic Aperture Radar (SAR) Support Data Extensions (SDE) for the National Imagery Transmission Format (Version 2.0) of the National Imagery Transmission Format Standard, Version 0.9, 20 May 1996.
VIMAS	Visible, Infrared, and Multispectral Airborne Sensor Support Data Extensions for the National Imagery Transmission Format (NITF) of the National Imagery Transmission Format Standard, 25 September 1997.

(Requests for copies of the above NIMA Specifications and Publications may be made to the National Imagery and Mapping Agency, Attn: NIMA/SESM, MS-D86, 4600 Sangamore Road, Bethesda, MD 20816-5003).

2.3 Standardized NATO Agreements

STANAG 4545	NATO Secondary Imagery Format (Version 1.0); Ratification Draft 1.

(Requests for copies of the above STANAG may be made to SAF/AQIJ, 1060 AF Pentagon (5D156), Washington, DC 20330-1060).

2.4 International Standards

ISO/IEC 12087-5: DIS	Information Technology; Computer graphics and image processing; Image Processing and Interchange; Functional Specification - Part 5: Basic Image Interchange Format.
ISO/IEC Directives	Procedures for the technical work of ISO/IEC JTC1 on Information Technology, Third Edition 1995.

ISO/IEC TR10000-1:1992	Information Technology - Framework and Taxonomy of International Standardized Profiles - Part 1: General principles and documentation framework, Third Edition, 1995.
ISO/IEC TR10000-2:1992	Information Technology - Framework and Taxonomy of International Standardized Profiles - Part 2: Principles and Taxonomy for OSI Profiles, Third Edition.
ISO/IEC 8632-1:1994	Information Technology - Computer Graphics Metafile for the Storage and Transfer of Picture Description Information - Part 1: Functional Specification.
ISO/IEC 8632-3:1994	Information Technology - Computer Graphics Metafile for the Storage and Transfer of Picture Description Information - Part 3: Binary Encoding.
ISO/IEC 8632:1992	Information Technology - Computer Graphics Metafile for the Storage and Transfer of Picture Description Information, AMD.1:1994 - Parts 1-4: Rules for Profiles.
ISO/IEC 10918-1:1994	Information technology - Digital Compression and Coding of Continuous-Tone Still Images: Requirements and Guidelines.
ISO/IEC 10918-2:1995	Information Technology - Digital Compression and Coding of Continuous-Tone Still iImages: Compliance Testing.
ISO/IEC 10918-3:DIS	Information Technology; Digital Compression and Coding of Continuous-Tone Still Images; Part 1: Extensions.
ISO/IEC 10918-4:DIS	Information Technology; Digital Compression and Coding of Continuous-Tone Still Images: Part 4; Registration Procedures for JPEG Profile, APPn Marker, and SPIFF Profile ID Marker.
ISO/IEC 9973:1994	1st Edition, Procedures for Registration of Graphical Items.
ISO/IEC 11072:1993	Information Technology - Computer Graphics - Computer Graphics Reference Model.

ISO/IEC 12087-1:1995	Information Technology - Computer Graphics and Image Processing - Image Processing and Interchange--Functional specification Part 1: Common architecture for imaging.
ISO/IEC 12087-2:1994	Information Technology - Computer Graphics and Image Processing - Image Processing and Interchange--Functional specification Part 2: Programmer's imaging kernel system application program interface.
ISO/IEC 12087-3:1995	Information Technology - Computer Graphics and Image Processing - Image Processing and Interchange--Functional specification Part 3: Image Interchange Facility (IIF).
ITU T.4 (1993:03)	Terminal Equipment and Protocols for Telematic Services - Standardization of Group 3 Facsimile Apparatus for Document Transmission, AMD2 08/95.

(Application for copies may be addressed to the American National Standards Institute, 13th Floor, 11 West 42nd Street, New York, NY 10036).

2.5 Other References

NITFS Tag Registry	NITFS Tagged Extensions Registry, latest update as posted at http://jitc-emh.army.mil/nitf/tag_reg/mast.htm.

3.0 OVERVIEW

3.1 Military and ISO Bandwidth Compression Standards

In the past, the NITFS has referenced Military Standards for it bandwidth compression requirements to ensure the interoperability of image files that were exchanged within the Intelligence Community (IC) and the Department of Defense (DOD) as well as support user requirements for compression of imagery. As new compression technologies were established within the commercial and government communities, they have been standardized within the DoD as Military Standards, or in commercial fora as ISO standards, and then profiled in and documented as Military Standards. An example of this is the Joint Photographic Experts Group (JPEG) standard, ISO/IEC 109018-1, which was adopted by the ISO/JTC1/SC 29 organization. Because the standard was written to support a great variety of uses (commercial imaging, medical imaging, digital cameras, etc.) it provided for capabilities that were not specifically required or useful to the NITFS user. A profile of the standard was written and documented in MIL-STD-198-199A, JPEG for the NITFS. This profile “scoped” the ISO standard by minimizing options, focusing parameters, and disallowing features that were not necessary for performance or interoperability. As there was no identified process on how to create or “register” profiles of the JPEG ISO standard at that time, it was documented as a Military Standard.

With the growing movement to migrate, where possible, from Military Standards to commercial standards, there has been a great deal of activity within the USIGS community to migrate the NITF and its associated standards completely to profiles of ISO standards (see NITFS Program Plan). For example, the NITF 2.1 format is in the process of becoming an ISO standard under the title Basic Image Interchange Format (BIIF), ISO/IEC 12087-5 DIS;

With respect to the NITFS compression standards, the JPEG Military Standard will be superceded by an “NITFS profile” of the ISO JPEG standard. Although almost identical technically to the Military Standard, this profile will be registered and approved by the ISO JTC1 organization as opposed to being a DOD standard. As an internationally approved profile, other organizations or entities, such as NATO, may choose to implement the profile, hence improving interoperability.

Within the ISO organization responsible for the development of Still Image compression standards, ISO JTC1/SC29/WG1, there are several activities currently underway which are of great interest to the imagery community. These activities are summarized in Table 3-1 below.

Table 3-1 NITFS Standards Transition Table

Work Item / Standard Name	Current Status	Anticipated Date for DIS	Remarks / Impact to the USIGS
"JPEG 2000"	Technology submissions, review and convergence (Mar 97-Mar 98)	Mar 2000	Will improve the compression and through-put by 10-20%
JPEG Multicomponent	Development of Committee Draft (Nov 97-Mar 98)	Jul 1999	Will allow for efficient dissemination and storage of multispectral data

3.1.1 JPEG 2000

JPEG 2000 is the title given to the follow-on to the currently defined JPEG standard, but which will most likely be a wavelet based solution. A key feature of this compression is that it will be based on a "modular" architecture framework. This facilitates insertion of new technologies in the future, provides for flexibility, and facilitates the potential to "swap" modules based on compression requirements (quality, rate, etc.) of individual users.

There is currently a "Call for Contributions" process, "whose goal is to: gather algorithms, components of algorithms, and architectural frameworks; and to organize algorithm components into a single architecturally based standard. An architecturally based standard has the potential of allowing the JPEG 2000 standard to evolve and integrate new algorithm components without requiring a new standards definition." [1]. Once all contributions are evaluated based on an available set of criteria, actual technical development of the standard begins.

A few additional requirements of this new algorithm include:

- Improved performance (greater compression rates)
- Improved image quality
- Flexibility to support different types of imagery (visible, IR, Multicomponent, etc.)
- Ability to support tiling, and very small and large sized imagery

[1] Call for Contributions for JPEG 2000 (JTC 1.29.14,15444): Image Coding System; International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC) Joint Technical Committee 1 /SC29/WG1 Document N505, 21 March 1997; page 4

The current schedule of activities for JPEG 2000 is provided below:

• Submission of algorithm contributions	Sep 97
• *Submission of architecture contributions	Oct 97
• Second experimental results and convergence	Mar 98
• WD to Committee Draft (CD)	Jul 98
• CD to final CD	Mar 99
• Submit CD for Draft International Standard (DIS)	Nov 99
• DIS submitted for International Standard (IS)	Mar 00
• IS	Nov 00

Profile development of the JPEG 2000 standard could potentially begin once it is accepted as a Draft International Standard (DIS).

JPEG 2000 potentially offers a number of benefits to the NITFS community, over a number of the existing compression algorithms within the USIGS, some of which are based on ISO standards, and others which are DOD specific. The varying techniques and performance of these compression protocols may have a negative impact to interoperability within the USIGS. Some issues of concern are:

- Current quality “hits” caused by multiple compression algorithms being used throughout the USIGS dissemination chain.
  - Images sent to a tactical unit will go through the 4.3 DPCM, 1.3 DCT and at least one JPEG compression, which can cause significant quality loss due to interactions of the compression algorithms and artifacts.
- Several organizations and systems have their own standard that meet their requirements but may not meet other system’s requirements.
  - The SRG 30 study started to quantify some of the loss expected by the different compression algorithms, and in multiple dissemination.
- The USIGS will become more of a database and “pull” environment than the current “push” environment
  - Current standard compression algorithms are not optimal for this environment.

JPEG 2000, which is based on the most current “state-of-the-art” technology, may be able to address many of these issues. Several studies show that advanced compression algorithms can improve performance throughout the USIGS. One of these algorithms, for example, is the Wavelet TCQ, which, in evaluations has done extremely well and has demonstrated a much improved bit rate and image quality capability than current algorithms. (Wavelet TCQ improved compression up to 30%.)

3.1.2 Multi-Component JPEG

The purpose of this Multi-Component JPEG standard, under the leadership of the ISO JTC1 /SC29/WG1 organization, is to provide a standard means of compressing and decompressing multiple-component, continuous tone images, in such a way that the reconstructed output has minimal image quality loss with respect to the original image. This standard would be applicable to those users who have imagery that does not subscribe well to the standard color compression techniques commonly used with the current JPEG, such as multispectral imagery, medical imagery (MRI, CAT scan), and color imagery.

With the proliferation of multiple new sources of high quality multispectral data, there is a need for the capability to store and disseminate this data in an efficient manner.

A primary goal for this algorithm is to maintain compatibility with the procedures defined in IS 10918-1 in order to maintain some level of backward compatibility to the current JPEG standard. There is an additional objective of developing in such that it will fit into the JPEG 2000 architecture framework, and hence, avoid the potential problem of having two standards that can potentially support multispectral imagery.

Below is the schedule for this work item;

- Submission of algorithm contributions Nov 96
- Development of WD Nov 97
- Development of CD Mar 98
- Development of DIS Jul 98
- IS Nov 98

The intention is, that once the Multi-Component JPEG Standard is approved as a DIS, a profile will be written such that it can be implemented by NITF/BIIF systems.

3.2 Bandwidth Compression for the NITFS

The National Imagery Transmission Format Standard (NITFS) is the standard for formatting digital imagery and geospatial information and related products and exchanging them among members of the Intelligence Community (IC) as defined by Executive Order 12333, the Department of Defense (DOD), and other departments and agencies of the United States Government, as governed by Memoranda of Agreement (MOA) with those departments and agencies . It is the required format for the exchange of digital imagery within the USIGS. A programmatic overview of the NITFS, including its history and strategic outlook, is documented in the NITFS Program Plan (NIMA NNPP).

An inherent component of the NITFS is the requirement for bandwidth compression of imagery using standard algorithms and technologies. As the NITFS has evolved, compression technologies, and the standards they have produced, have evolved tremendously. To ensure interoperability, many of these “older” standards have been retained as requirements under the NITFS for backwards compatibility, although new standards have been introduced, and have slowly superceded those not meeting the USIGS user requirement as well.

Briefly described in this section are the standards currently mandated by the NITFS. Detailed technical specifications of these standards are provided in this document if the standard is a USIGS profile of an ISO standard, or are referenced in related NITFS documents if the standards are already documented existing Military Standards.

3.2.1 Adaptive Recursive Interpolated Differential Pulse Code Modulation (ARIDPCM) Image Compression for the NITFS

The ARIDPCM standard was required in the previous versions of NITFS, prior to the establishment of JPEG. In the NITF 2.1 timeframe, requirements to “pack” or “unpack” ARIDPCM have been removed. Because JPEG provides for a much improved image quality and compression rate, it is expected that the only files containing ARIDPCM compressed imagery will be from legacy systems, archives and databases. These files will be converted to JPEG for interoperability reasons.

ARIDPCM is documented in MIL-STD-188-197A. The DISA CFS is responsible for the administrative management of the standard; NIMA SES is responsible for its technical development.

3.2.2 Bi-Level Image Compression for the NITFS

This standard establishes the requirements to be met by NITFS systems when image data are compressed using the bi-level facsimile compression specified by the International Telecommunications Union (ITU) International Telegraph and Telephone Consultative Committee (CCITT) Recommendation T.4 and MIL-STD-188-161C for Group 3 facsimile devices.

It is profiled in MIL-STD-188-196 for the NITFS, and includes additional implementation notes and guidelines in addition to what is provided within the ITU standard.

3.2.3 Joint Photographic Experts Group (JPEG) Image Compression for the NITFS

This standard, MIL-STD-188-198, establishes the requirements to be met by systems complying with NITFS when image data are compressed using the JPEG image compression algorithm as described in ISO/IEC 10918-1*, Digital Compression and Coding of Continuous-tone Still Images*. The highly successfully JPEG standard provides technical detail of Lossless and Lossy compression algorithms, although only the Lossy algorithm is implemented within the MIL-STD-188-198, for both 8- and 12-bit gray scale imagery and 24-bit color imagery.

3.2.4 Vector Quantization Decompression for the NITFS

MIL-STD-188-199, Vector Quantization Decompression for the National Imagery Transmission Format Standard, 27 June 1994, establishes the requirements to be met by NIFTS-compliant systems when image data are decompressed using the Vector Quantization (VQ) compression algorithm. This allows NITFS-compliant systems to accept and decompress data that are compressed using a VQ compression scheme. Commonly used for NIMA maps, this standard describes the VQ compression in the general requirements section, but does not fully describe the steps for compression. It fully describes the steps involved in decompressing VQ compressed images.

3.3 Summary

Table 3-2 below provides a summary overview of the compression techniques addressed in this document and their application.

Table 3-2 Compression Technique Overview

Compression Technique	Documentation	Guidance/Usage
ARIDPCM	MIL-STD-188-196	Only to be used for decompression of legacy NITFS 1.1 imagery and databases.
Bi-Level	MIL-STD-188-197 ITU-TRecom. T.4	Used to compress/decompress bi-level imagery and graphic overlays and to transcode with compressed G-III fax data.
JPEG Lossy	MIL-STD-188-198A ISO/IEC 10918-1,-2,-3	Is the primary compression of gray-scale imagery within the NITFS. The JPEG DCT can compress imagery that is either 8 or 12 bpp and performs best in the range of 4.0 bpp to 0.5 bpp with optimized tables that are supplied by NITFS.
NITFS VQ	MIL-STD-188-199	Is used to decode VQ compressed data. The main application is the decoding of geo-spatial data (i.e., maps).
NIMA Interpolation Method 4	NIMA N-0106-97	Is used as a preprocessor to allow for the compression of grayscale imagery to very low bit rates. This is used in conjunction with the JPEG DCT algorithm to improve the compression at very low bit rates (0.5 bpp to 0.0625 bpp)
JPEG Optimized tables	NIMA N-0106-97	Quantization tables and Huffman tables optimized for image type (IR, Visible, SAR, Other) and desired bit rate for the JPEG DCT, are supplied by NIMA.
Pre- Post-Processing	NIMA N-0106-97	Pre- and Post-processing techniques are used to modify the data before and after compression to improve the quality. This is only used within the primary dissemination systems for high-quality compression.
JPEG Lossless	ISO/IEC 10918-1,-2,-3, -4	JPEG lossless is used when there is a desire to have no numerical loss of data. This BWC technique can compress imagery from 4 to 16 bits per pixel data and achieves only 2:1 compression.

Table 3-2 Compression Technique Overview (continued)

Compression Technique	Documentation	Guidance/Usage
Multiple Component JPEG	New ISO documentation	This is a new development of the ISO JPEG group to improve the compression of multiple component data (i.e., multispectral). This should allow for the efficient dissemination and storage of multispectral data.
JPEG 2000	New ISO documentation	This is a new compression standard, expected in 2000, which should replace all other compression algorithms within the USIGS and promote better throughput and interoperability.

Table 3-3 NITFS Profiles to International Standards / Profile

Standard Name	Current NITFS Profile	Anticipated International Standard / Profile	Expected Approval of Profile Date	Anticipated Impact to Existing Implementation
JPEG (lossy) ISO/IEC 10918-1	MIL-STD-188-198	ISO 10918-1 profiles	TBR	None/Minor
JPEG Lossless ISO/IEC 10918-1	N/A	ISO 10918-1/4 profiles	TBR	New Capability
CCITT Recommendation T.4	MIL-STD-188-196	CCITT Recommendation T.4	TBR	None/Minor
Vector Quantization	MIL-STD-188-199	ISO 12087-5 ISP	TBR	None/Minor
ARIDPCM (Legacy Only)	MIL-STD-188-197	N/A (Legacy support ONLY)	TBR	N/A

4.0 JPEG LOSSY COMPRESSION

This standard establishes the requirements to be met by systems complying with the NITFS when image data are compressed using the JPEG image compression algorithm as described in DIS 10918-1*, Digital Compression and Coding of Continuous-tone Still Images*. The requirements specified in the NITFS JPEG profile are intended to enable the interchange of 8- and 12- bit gray scale imagery and 24-bit color imagery compressed with JPEG. As part of the migration to international standards a USIGS profile of the Lossy JPEG standard, technically identical to MIL-STD-188-198 will supersede the Military Standard.

Refer to MIL-STD-188-198A, CN2, for the detailed implementation of the standard.

5.0 DOWNSAMPLE JPEG COMPRESSION (NIMA METHOD 4)

The specification for downsample JPEG is the standardized result of field trials of the approach also known as “NIMA Method 4.” The NIMA Method 4 approach provides a means to use existing lossy JPEG capabilities in the field to get increased compression for use with low bandwidth communications channels. This gives the field a very cost-effective approach for a critical capability during the period that the JPEG 2000 solution is being resolved. NIMA Method 4 specifically correlates to a selection option (Q3) within downsample JPEG that provides a very useable tradeoff between file compression and the resulting loss in quality.

5.1 General Requirements

5.1.1 Interoperability

This profile is intended to enable the interchange in the NITFS format, of 8-bit (Type 1) and 12-bit (Type 2) gray scale imagery compressed with the downsample JPEG algorithm. This algorithm is based on the JPEG sequential DCT image compression algorithm as described in ISO 10918-1*, Digital Compression and Coding of Continuous-tone Still Images* and the NITFS implementation of JPEG described in MIL-STD-188-198A, *Joint Photographic Experts Group (JPEG) Image Compression for the National Imagery Transmission Format Standard*. The algorithm uses a specified downsampled filter prior to JPEG sequential DCT compression and a specified upsampling filter following JPEG decompression. This profile establishes the requirements for the communication or storage for interchange of image data in compressed form. Each type of operation defined by this profile consists of three parts:

- The compressed data intechange format (which defines the image data field of the NITF file format
- The encoder
- The decoder

This profile defines two types of operation:

- Type 1: 8-bit sample precision gray scale sequential Discrete Cosine Transform (DCT) with Huffman coding.
- Type 2: 12-bit sample precision gray scale sequential Discrete Cosine Transform (DCT) with Huffman coding.

5.1.2 Encoders

Encoders shall output to the image data field of the NITF file a full interchange format that includes the compressed image data and all table specifications used in the encoding process as illustrated in Figure 5-1.

5.1.2.1 Image downsampling

The downsample JPEG algorithm encoder utilizes a downsampling procedure to extend the low bit-rate performance of the NITFS JPEG algorithm described in MIL-STD-188-198A. Figure 5-2 illustrates the concept. The downsampling preprocessor allows the JPEG encoder to operate at a higher bit-rate on a smaller version of the original image while maintaining an overall bit-rate that is low.

5.1.2.2 JPEG encoding

Once downsampling of the original image is performed, the encoding process is identical to that of NITFS JPEG lossy compression algorithm. Minor variations exist in the compressed data format as described in this document. The NITFS JPEG algorithm is a profile of the lossy DCT-based encoding algorithm found in ISO 10918-1. Encoders conforming to this profile may use any procedure in ISO 10918-1 applicable to DCT encoding subject to the requirements and restrictions expressed in MIL-STD-188-198A and herein.

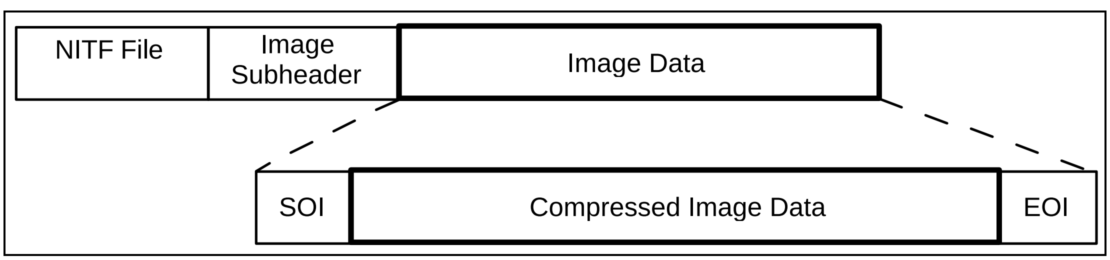

FIGURE 5-1 NITF FILE STRUCTURE

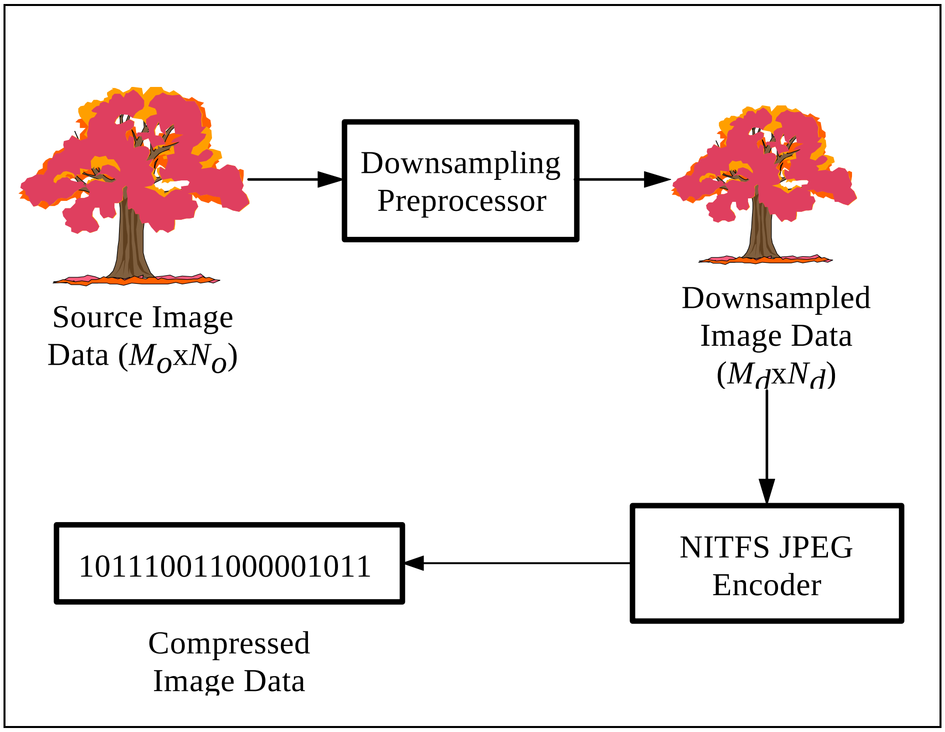

FIGURE 5-2 DOWNSAMPLE JPEG ALGORITHM ENCODER

5.1.3 Decoders

All decoders shall interpret both the full interchange format and the abbreviated interchange format.

5.1.3.1 JPEG decoding

The downsample JPEG algorithm decoder decodes the compressed image data using the NITFS JPEG decoder (see Figure 5-3). This results in reconstructed image data whose dimensions match that of the downsampled image data in Figure 5-2.

5.1.3.2 Image upsampling

An upsampling postprocessor is used to return the reconstructed image data to the same dimensions as that of the original. It is important to note that the down/ upsampling processes are not lossless. The downsample JPEG algorithm makes a tradeoff between JPEG and down/upsampling artifacts in the reconstructed imagery.

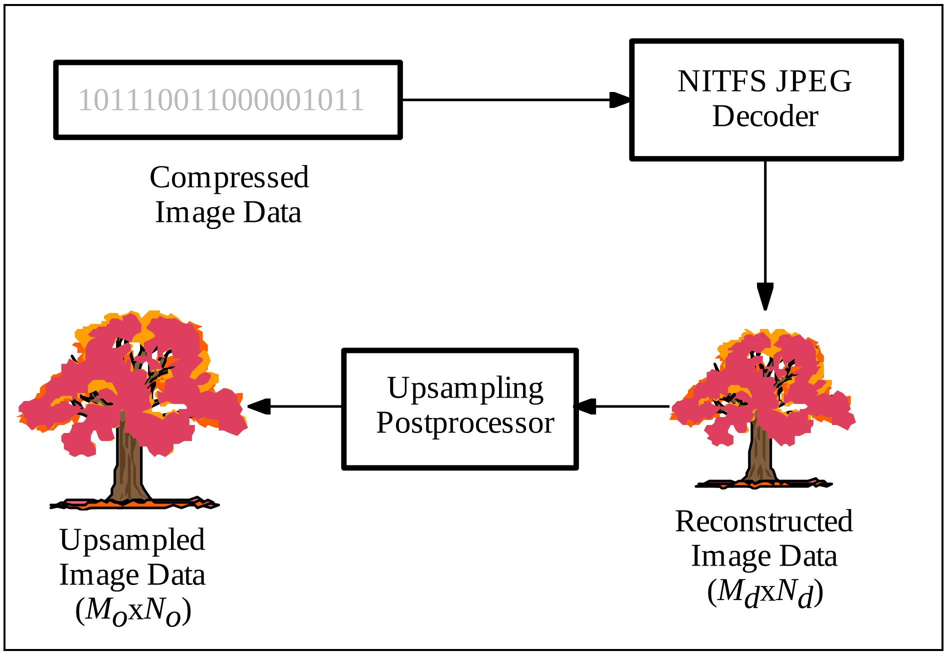

FIGURE 5-3 DOWNSAMPLE JPEG ALGORITHM DECODER

5.1.4 Interchange format-encoders

Encoders shall output to the image data field of the NITF file either a full or abbreviated interchange format. The full interchange format includes the compressed image data and all table specifications used in the encoding process.

In MIL-STD-188-198A, an abbreviated interchange format is defined that is identical to the full interchange format, except that it does not contain all tables required for decoding a compressed image data file. This capability allows for smaller files, but requires continual maintenance and dissemination of prepositioned default tables which can not be reliably achieved. Currently, non embedded default tables are permitted by this profile. However, future NITF compliant systems will be required to embed all necessary tables in the compressed data stream. Implementors are strongly encouraged to avoid usage of default tables.

The tables given in Appendix B of this document are recommended tables. The visible imagery 8- and 12-bit tables in MIL-STD-188-198A are allowed as defaults. Applications are free to develop tables more appropriate to their imagery than those described here. Any such custom tables must be embedded in the data stream.

5.1.5 Further general requirements

Further requirements regarding the NITFS JPEG lossy compression algorithm apply to this profile and may be found in MIL-STD-188-198A. In the event of a conflict between the text of this profile and MIL-STD-188-198A, this profile shall take precedence.

5.2 Detailed Requirements

5.2.1 Image downsampling process

Downsampling is the process of reducing the size of an image relative to the original. In this document, downsampling is performed by simultaneously filtering the image and selecting a subset of the total samples available from the original. The output image will have fewer pixels and reduced dimensions, but will still be recognizable as the original image. Furthermore, the downsampling is carried out such that the aspect ratio is consistent between the original and downsampled images. Inputs into the downsampling module are the original image and a downsample ratio that relates the total number of pixels in the original image to the desired downsampled image. The output of this module is delivered to a NITFS compliant JPEG module for further compression and formation of a coded bitstream.

The calculation of the downsampled image dimensions and adjusted downsample ratios are discussed in Section 5.2.1.1. The mechanics of the one-dimensional filtering operation are explained in Section 5.2.1.2, while the necessary equations to calculate the filter parameters are given in Section 5.2.1.2.1. Section 5.2.1.3 describes in general how the filtering operation is applied to images.

5.2.1.1 Downsampled image dimensions

The downsampled image will have reduced dimensions with respect to the original image. The number of rows and columns of the downsampled image must be calculated separately in order to properly account for non-square images. Since the downsampled image data will be compressed with JPEG, greater coding efficiency can be obtained by tuning the downsampling ratio to create downsampled image dimensions which are integer multiples of 8. This prevents the JPEG algorithm from padding blocks at image edges and wasting bit rate. The following equations calculate the proper downsampled image dimensions for maximum JPEG coding efficiency:

Downsampled image rows: $N_d = 8 \cdot round\left(\frac{N_o}{8 \cdot \sqrt{R_o}}\right)$

Downsampled image columns: $M_d = 8 \cdot round\left(\frac{M_o}{8 \cdot \sqrt{R_o}}\right)$

where N_o and M_o are the number of rows and columns in the original image respectively, and R_o is the original downsample ratio. The actual downsample ratio that is to be used in subsequent processing should be altered to reflect the fact that truncation is performed to obtain the downsampled image dimensions. For this situation, the downsample ratio must be calculated separately for the row and columns dimensions.

Downsample ratio for the row dimension:

$$R_{row} = \frac{N_o}{N_d}$$

Downsample ratio for the column dimension:

$$R_{col} = \frac{M_o}{M_d}$$

5.2.1.2 Downsampling filter operation

The downsampled image is formed by performing separable one-dimensional filtering on the rows and columns of the original image. The filtering operation is described in the following equation as the weighted average of samples.

Filtering equation for downsampling:

$$y_i = \sum_{j=b_i}^{e_i} w_{ij} \cdot x_j$$

where y_i denotes a sample in the output image, x_j denotes a sample in the input image, b_i and e_i specify integer limits on the summations, and w_{ij} is the filter coefficient associated with output sample, i, and input sample, j. When filtering is performed in the row dimension, then y_i and x_j refer to row samples; when filtering is performed in the column dimension, then y_i and x_j refer to column samples. The equation is applied similarly for all elements in a single dimension using the same set of parameters, e_i, b_i, and w_{ij}, so no designation has been made for the particular row or column that is filtered. However, the integer limits and the filter coefficients must be calculated separately for the rows and columns when the image is non-square. The filtering operation is illustrated in Figure 5-4 for the row processing case with i = 10, b_{10} = 11, e_{10} = 15, and a downsample ratio, R = 1.3.

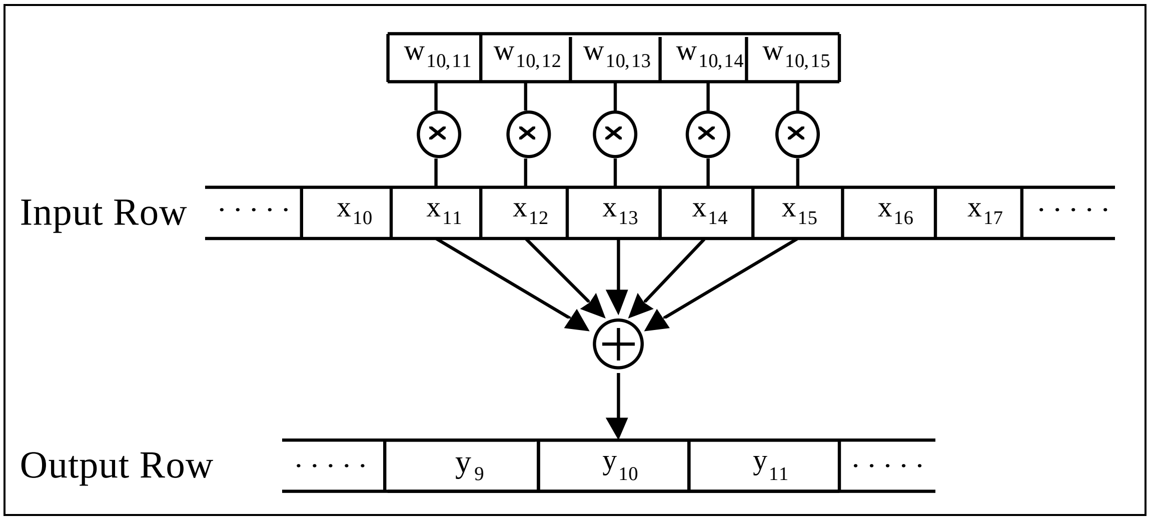

FIGURE 5-4 DOWNSAMPLING DEMONSTRATION WITH I=10, B_{10}=11, E_{10}=15, AND R = 1.3

5.2.1.2.1 Downsample filter parameter calculations

The pertinent parameters that are required for implementation of the downsampling filter are the integer summation limits, b_i and e_i, and the coefficients, w_{ij}. The calculation of these parameters differs slightly depending on the dimension that is considered, due to the change in downsample ratio as discussed in Section 5.2.1.1. However, one set of parameters can be used for all the elements in the associated dimension (e.g. one set of row parameters can be applied to all the rows).

The summation limits can be calculated as shown in the following equations.

Filter beginning index: $b_i = ceil\left(\ _i - (\ \cdot R)/2\right)$

Filter ending index: $e_i = floor\left(\ _i + (\ \cdot R)/2\right)$

where:

$$= 4$$

$$R = \begin{cases} R_{row} & \text{for row processing} \\ R_{col} & \text{for column processing} \end{cases}$$

$${}_i = i \cdot R + 0.5 \cdot R - 0.5$$

The parameter, *a*, is a value that specifies a fixed filter length, while *R* refers to the downsample ratios discussed in Section 5.2.1.1. b_i is a variable describing the location of the filter center relative to the input samples.

The filter coefficients, w_{ij}, can be calculated in a two-step process.

Filter coefficients:

$$w_{ij} = \frac{c_{ij}}{\sum_{j=b_i}^{e_i} c_{ij}}$$

where:

$$c_{ij} = \sqrt{\cos\left(\frac{\cdot({}_i - j)}{\cdot R}\right)} \times \mathrm{sinc}\left(\frac{\cdot({}_i - j)}{R}\right)$$

and:

$$\mathrm{sinc} = \begin{cases} \dfrac{\sin(x)}{x} & x \neq 0 \\ 1 & x = 0 \end{cases}$$

sinc(*x*)

5.2.1.3 Application of the downsampling filter

One-dimensional filtering is applied repeatedly along each dimension until all samples in the downsampled image have been computed. Filtering along each dimension is performed independently. One dimension is processed entirely before continuing to the complementary dimension. After processing one dimension, an intermediate image is formed as the input for processing in the other dimension. Note that the processing order (e.g. rows then columns, or vice versa) can be chosen so as to maximize performance for a given system platform. These concepts are further described in Figure 5-6 which shows the control procedure for the downsampling operation for the example of row-column order processing.

5.2.1.3.1 Downsampling along the image edges

In the course of downsampling an image, input values are needed that lie outside the original image. This occurs at the top, bottom, left, and right edges of the image. When extra data is needed, enough samples shall be generated by mirroring values from within the image so that the filter coefficients will always coincide with actual image samples. The mirroring point coincides with the input data sample that is exactly on the edge (e.g. first sample in a row when padding on the left of the image). Therefore, the edge sample is never repeated. This is illustrated in Figure 5-5.

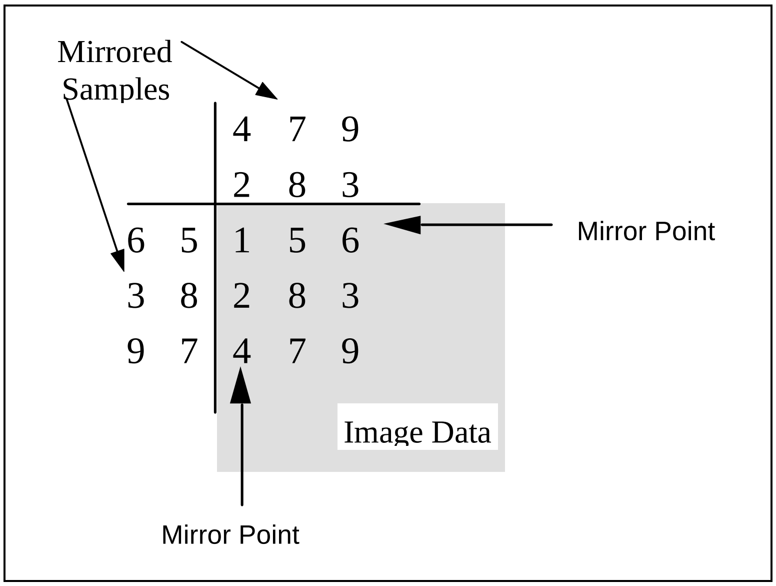

FIGURE 5-5 ILLUSTRATION OF MIRRORING FOR IMAGE EDGES

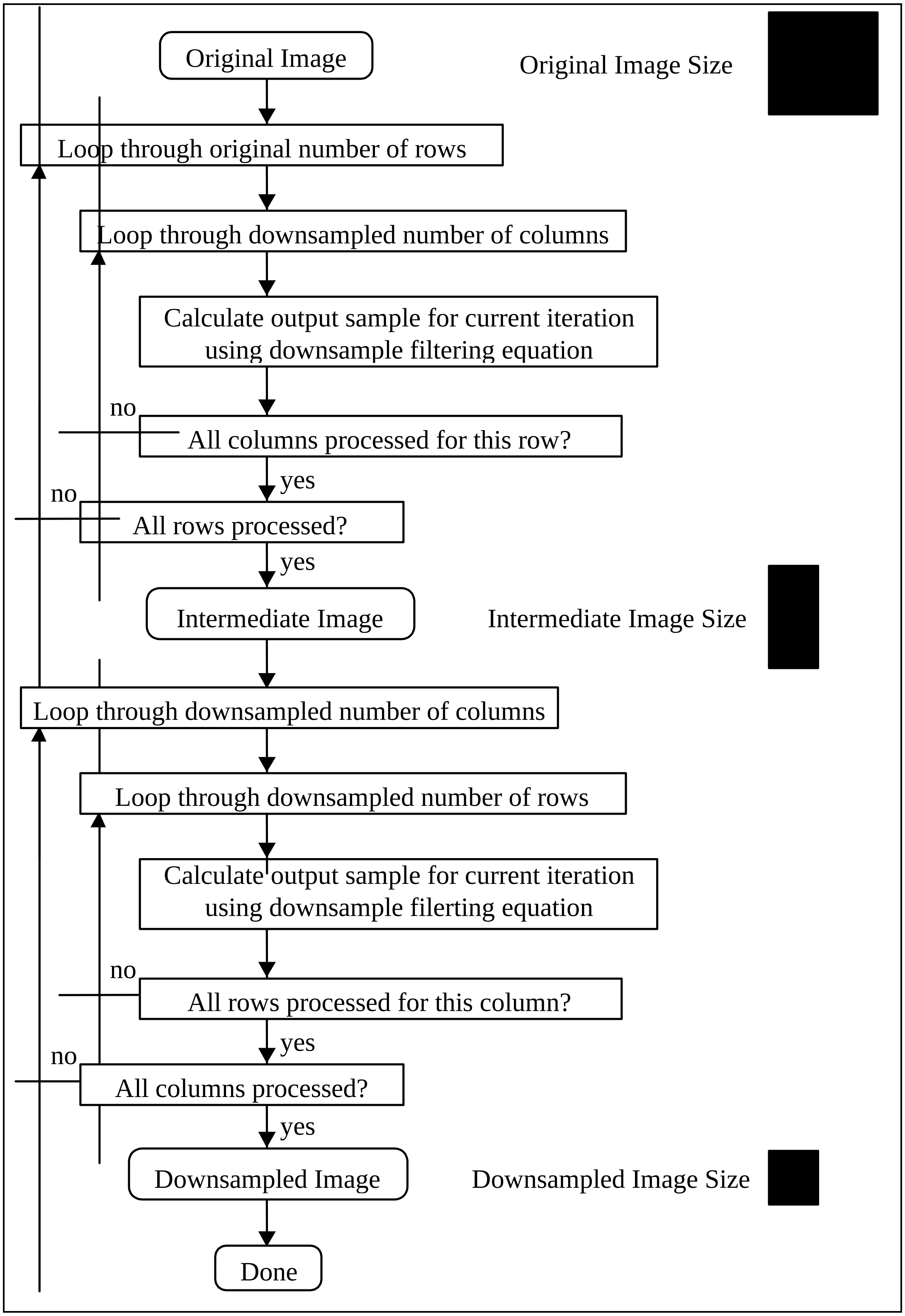

FIGURE 5-6 CONTROL PROCEDURE FOR IMAGE DOWNSAMPLING (ROW-COLUMN ORDER)

5.2.2 JPEG compression of the downsampled image

The requirements and control procedures pertaining to the sequential DCT-based JPEG mode in MIL-STD-188-198A apply to this profile except as noted below. Once downsampling of the original image data is completed, the resultant downsampled image data is compressed with the NITFS JPEG sequential DCT lossy mode image compression algorithm. Changes in the compressed image data format are described in the following sections. The appropriate flags and parameter values for relevant fields in the NITF image subheader are given for the downsample JPEG algorithm in Section 5.2.2.2.2. Suggested Quantization and Huffman Tables for both 8-bit and 12-bit gray scale imagery may be found in Appendix B.

5.2.2.1 Control procedures for the sequential DCT lossy mode

The control procedures for encoding an image using the JPEG sequential mode may be found in ISO 10918-1. It is required by this profile that an NITF APP_6 “NITF” application data segment be placed in the compressed data stream. This data segment immediately follows the first SOI marker in the Image Data Field (see Figure 5-7). The format and content of this data segment are discussed in Section 5.2.2.2.3. Additional requirements and control procedures for NITFS JPEG sequential DCT mode may be found in MIL-STD-188-198A.

5.2.2.2 Compressed data interchange format

The interchange format consists of an ordered collection of markers, parameters, and entropy-coded data segments. A detailed description of the format is given in MIL-STD-188-198A. The following sections provide the required changes that are necessary when using the downsample JPEG algorithm.

5.2.2.2.1 Format of a JPEG compressed image within an NITF file

The format for NITF image data compressed with the JPEG sequential DCT lossy mode differs based on the number of blocks, bands, and IMODE value B (see MIL-STD-2500A). These different cases are described below. Note that IMODE = S, and P are not appropriate for the down sampled JPEG algorithm since this profile is single band (8-bit and 12-bit gray scale) in nature.

5.2.2.2.1.1 Single block JPEG compressed format

The downsampled JPEG algorithm shall be limited to original image data no larger than 2048 by 2048 pixels, single block. The format for NITF single block image data compressed with the sequential lossy JPEG mode is shown in Figure 5-7

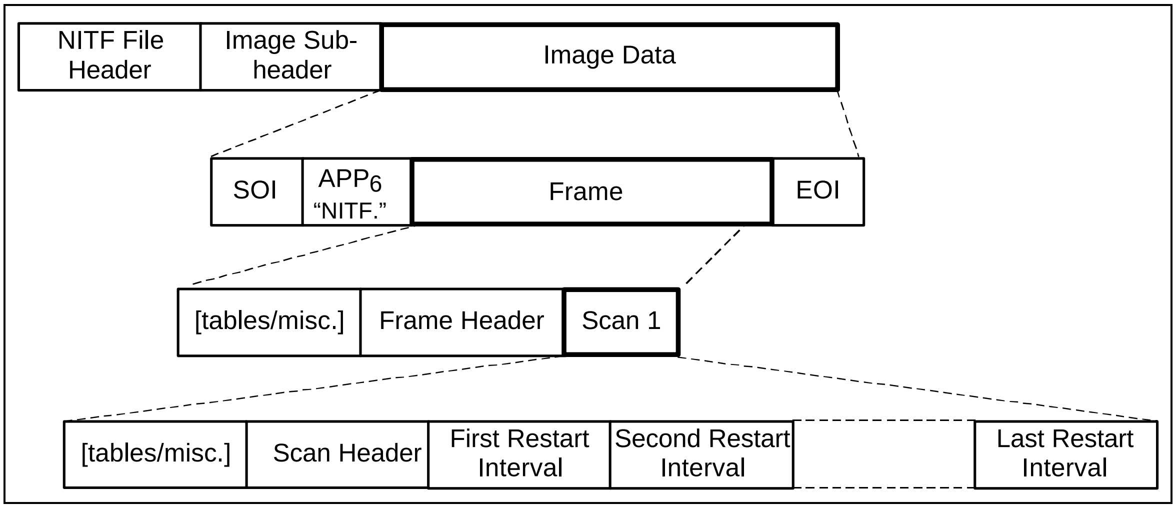

FIGURE 5-7 NITF SINGLE BLOCK FILE STRUCTURE (IMODE=B)

5.2.2.2.1.1.1 Single block image data format

The top level of Figure 5-7 specifies that the JPEG compressed data is contained in the Image Data Field of the NITF file. The second level of Figure 5-7 specifies that the single block image format shall begin with an SOI marker, shall contain one frame, and shall end with an EOI marker. Between the SOI/EOI marker pair, the data stream is compliant with ISO 10918-1 subject to the requirements and constraints of this profile.

5.2.2.2.1.1.2 Frame and Scan formats

The frame and scan marker formats in Figure 5-7 are the same as those found in MIL-STD -188-198A. The Start-of-Frame (SOF) marker segment contains two fields “Y” and “X” which contain the number of lines and the number of samples per line in the compressed image. For the downsample JPEG algorithm, these fields shall contain the number of lines and the number of samples per line for the downsampled image data. These fields must reflect the size of the image that underwent JPEG compression.

5.2.2.2.1.2 Multiple block JPEG compressed format

Downsampling JPEG shall not be used in conjunction with multiple blocked images. Such images must be converted to a single block, less than 2048 by 2048 pixels, then downsampled.

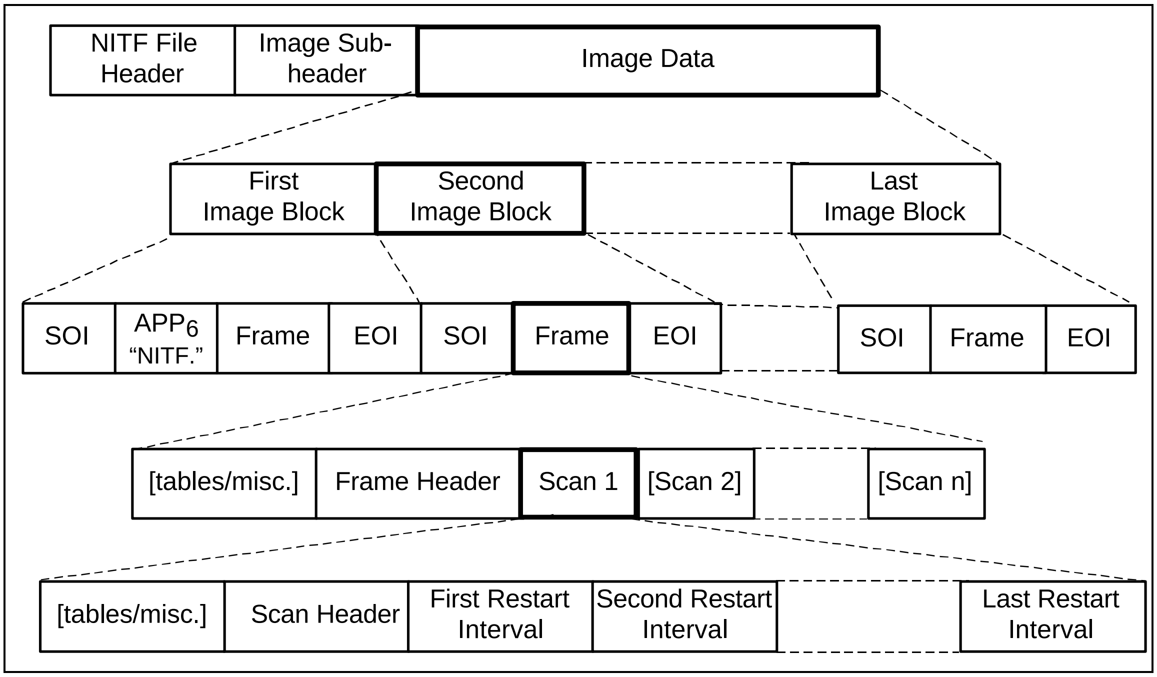

FIGURE 5-8 NITF MULTIPLE BLOCK FILE STRUCTURE (IMODE=B OR P)

5.2.2.2.2 NITF image subheader

Fields in the NITF image subheader must reflect the original image size. The downsample JPEG algorithm is unique in that the image and block sizes in the NITF image subheader do not match the image or block sizes in the JPEG SOF marker data segment(s). This is necessary since the JPEG compression operates on the downsampled image or blocks while ancillary NITF data such as overlays apply only to the original image or block dimensions. The IC field of the NITF image subheader is set to I1. This signals a decoder that a downsampled JPEG compressed image follows.

The NROWS and NCOLS fields of the image subheader shall contain the number of significant rows and columns, respectively, in the original image. The NPPBH and NPPBV fields shall contain the number of pixels per block horizontal and the number of pixels per block vertical, respectively, of the original blocks in a blocked image. Since downsampled JPEG will only be applied to single block images NPPBH will be equal to NCOLS and NPPBV will be equal to NROWS.

The IMAG field of the NITF image subheader is not modified for the downsample JPEG algorithm. Any decoder capable of decoding a downsampled JPEG compressed data file must restore the image and blocks to their original dimensions.

The COMRAT field of the NITF image subheader shall be set to 00.0 or 04.0. The 00.0 indicates that general purpose Huffman and Quantization Tables have been embedded into the JPEG stream. The 04.0 indicates that the specially developed tactical imagery Huffman and Quantization Tables have been embedded into the JPEG stream. These tactical tables can be found in Appendix B.

5.2.2.2.3 APP$_6$ “NITF” application data segment

NITF requires the use of an APP$_6$ “NITF” application data segment. This application data segment shall immediately follow the first SOI marker in the image data field. The “NITF” application data segment contains information which is needed by an interpreter but not supported by the ISO/CCITT JPEG format.

TABLE 5-I. APP_6 "NITF" Application Data Segment

Offset	Field Value	Field Name	length (bytes)	Comments
0	0xFFE6	APP_6	2	NITF application data marker.
2	25	L_p	2	Segment length (2+length of application data)
4	0x4E49 0x5446 0x00	Identifier	5	Null terminated string: "NITF"
9	0x0200	Version	2	Version number. The most significant byte is used for major revisions, the least significant byte for minor revisions. Version 2.00 is the current revision level.
11	0x42	IMODE	1	Image Format. 'B' - IMODE=B
12	0x0001	H	2	Number of image blocks per row.
14	0x0001	V	2	Number of image blocks per column.
16	0-00	Image Color	1	Original image color representation. One value is defined at this time. 0 - monochrome
17	0x08 and 0x0C	Image Bits	1	Original image sample precision.
18	0-99	Image Class	1	Image data class (0-99). One value is defined at this time 00 - general purpose 04 - tactical (downsampled) imagery
19	0x01 or 0x04	JPEG Process	1	JPEG coding process. The values for this field are defined to be consistent with ISO IS 10918-2. 1 - baseline sequential DCT, Huffman coding, 8-bit sample precision 4 - extended sequential DCT, Huffman coding, 12-bit sample precision
20	0x00	Quality	1	Image default Quantization & Huffman tables used. The value 0 indicates no defaults and all quantization tables must then be present in the stream.
21	0	Stream Colour	1	Compressed colour representation. One value is defined at this time. 0 - monochrome
22	0x08 or 0x0C	Stream Bits	1	COMPRESSED IMAGE SAMPLE PRECISION .
23	0	Flags	4	RESERVED FOR FUTURE USE .

5.2.3 JPEG decompression of the downsampled image

Prior to upsampling, JPEG decompression takes place resulting in a lossy reconstruction of the downsampled image data. The control procedures for decoding an image compressed with the JPEG sequential DCT lossy mode may be found in ISO 10918-1. These procedures are to be followed pursuant to the requirements of MIL-STD-188-198A.

5.2.4 Image upsampling process

Upsampling is the process of increasing the number of samples through interpolation of the existing values. The process is very similar to downsampling as described in Section 5.2.1, but in this case the image will be sampled more frequently to increase the number of image samples. Filtering is applied to the downsampled, reconstructed image that is received from the NITFS-compliant JPEG reconstruction module. The filtering operation generates enough new samples so that the upsampled image dimensions will match the dimensions of the original image.

Calculation of the upsample ratios is discussed in Section 5.2.4.1. The mechanics of the one-dimensional filtering operation are explained in Section 5.2.4.2, while the necessary equations to calculate the filter parameters are given in Section 5.2.4.2.1. Section 5.2.4.3 describes in general how the filtering operation is to be applied to images.

5.2.4.1 Upsample ratio calculation

Separate upsample ratios must be calculated for each dimension. The two upsample ratios define the amount of expansion that is required in order to match the resolution of the downsampled image to the original image. The ratios are only dependent on the number of rows and columns in the original image, specified by N_o and M_o, and the downsampled image, specified by N_d and M_d.

Upsample ratio for the row dimension:

$$R_{row} = \frac{N_o}{N_d}$$

Upsample ratio for the column dimension:

$$R_{col} = \frac{M_o}{M_d}$$

Note that the ratios are equivalent to the downsample ratios shown in Section 5.2.1.1.

5.2.4.2 Upsampling filter operation

The upsampled image is formed by performing separable one-dimensional filtering on the rows and columns of the downsampled image. The mechanics of the upsample filtering process is exactly the same as the downsample case with the exception of parameter calculation. The following equation is equivalent to the filtering equation for downsampling found in Section 6.2.1.2, but repeated here for convenience.

Filtering equation for upsampling:

$$y_i = \sum_{j=b_i}^{e_i} w_{ij} \cdot x_j$$

where y_i denotes a sample in the output image, x_j denotes a sample in the input image, b_i and e_i specify integer limits on the summations, and w_{ij} is the filter coefficient associated with output sample, i, and input sample, j. When filtering is performed in the row dimension, then y_i and x_j refer to row samples; when filtering is performed in the column dimension, then y_i and x_j refer to column samples. The equation is applied similarly for all elements in a single dimension using the same set of parameters, e_i, b_i, and w_{ij}, so no designation has been made for the particular row or column that is filtered. However, the integer limits and the filter coefficients must be calculated separately for the rows and columns when the image is non-square. The filtering operation illustrated in Figure 5-9 for the row processing case with i = 10, b_{10} = 6, e_{10} = 9, and an upsample ratio, R = 1.3.

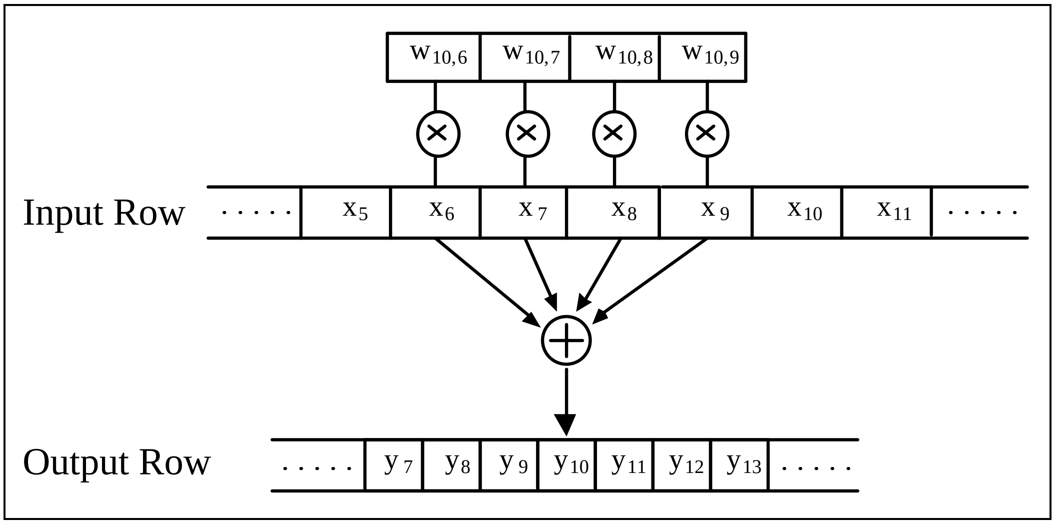

FIGURE 5-9 UPSAMPLING DEMONSTRATION WITH i=10, b_{10}=6, e_{10}=9, AND R = 1.3

5.2.4.2.1 Upsample filter parameter calculations

Similar to downsampling, the parameters that require calculation are the integer summation limits and filter coefficients. One set of parameters can be applied for all the elements in the associated dimension (e.g. one set of row parameters can be applied to all the rows).

Filter beginning index: $b_i = ceil\left(\ _i - \ /2 \right)$

Filter ending index: $e_i = floor\left(\ _i + \ /2 \right)$

where:

$$= 4$$

$$R = \begin{cases} R_{row} & \text{for row processing} \\ R_{col} & \text{for column processing} \end{cases}$$

$$_i = i \cdot R - 0.5 \cdot R + 0.5$$

The parameter, *a*, is a value that specifies a fixed filter length, while *R* refers to the upsample ratios discussed in Section 5.2.4.1. b_i is a variable describing the location of the filter center relative to the input samples. (Refer to Section 5.3 for more information concerning the upsample filter length).

The filter coefficients, w_{ij}, can be calculated in a two-step process.

Filter coefficients:

$$w_{ij} = \frac{c_{ij}}{\sum_{j=b_i}^{e_i} c_{ij}}$$

where:

$$c_{ij} = \left(\cos\left(\frac{\ \cdot (\ _i - j)}{\ } \right) \right)^2 \times \mathrm{sinc}\left(\ \cdot (\ _i - j)\right)$$

5.2.4.3 Application of the upsampling filter

One-dimensional filtering is applied repeatedly along each dimension until all samples in the upsampled image have been computed. Filtering along each dimension is performed independently. One dimension is processed entirely before continuing to the complementary dimension. After processing one dimension, an intermediate image is formed as the input for processing in the other dimension. Note that the processing order (e.g. rows then columns, or vice versa) can be chosen so as to maximize performance for a given system platform. These concepts are further described in Figure 5-10 which shows the general procedure for the upsampling operation for the example of column-row order processing.

5.2.4.3.1 Upsampling along the image edges

In the course of upsampling an image, input values are needed that lie outside the sampling grid of the downsampled image. This occurs at the top, bottom, left, and right edges of the image. When extra data is needed, enough samples shall be generated by mirroring values from within the image so that the filter coefficients will always coincide with actual image samples. The mirroring point coincides with the input data sample that is exactly on the edge (e.g. first sample in a row when padding on the left of the image). Therefore, the edge sample is never repeated. This is illustrated in Figure 5-5 found in Section 5.2.1.3.1.

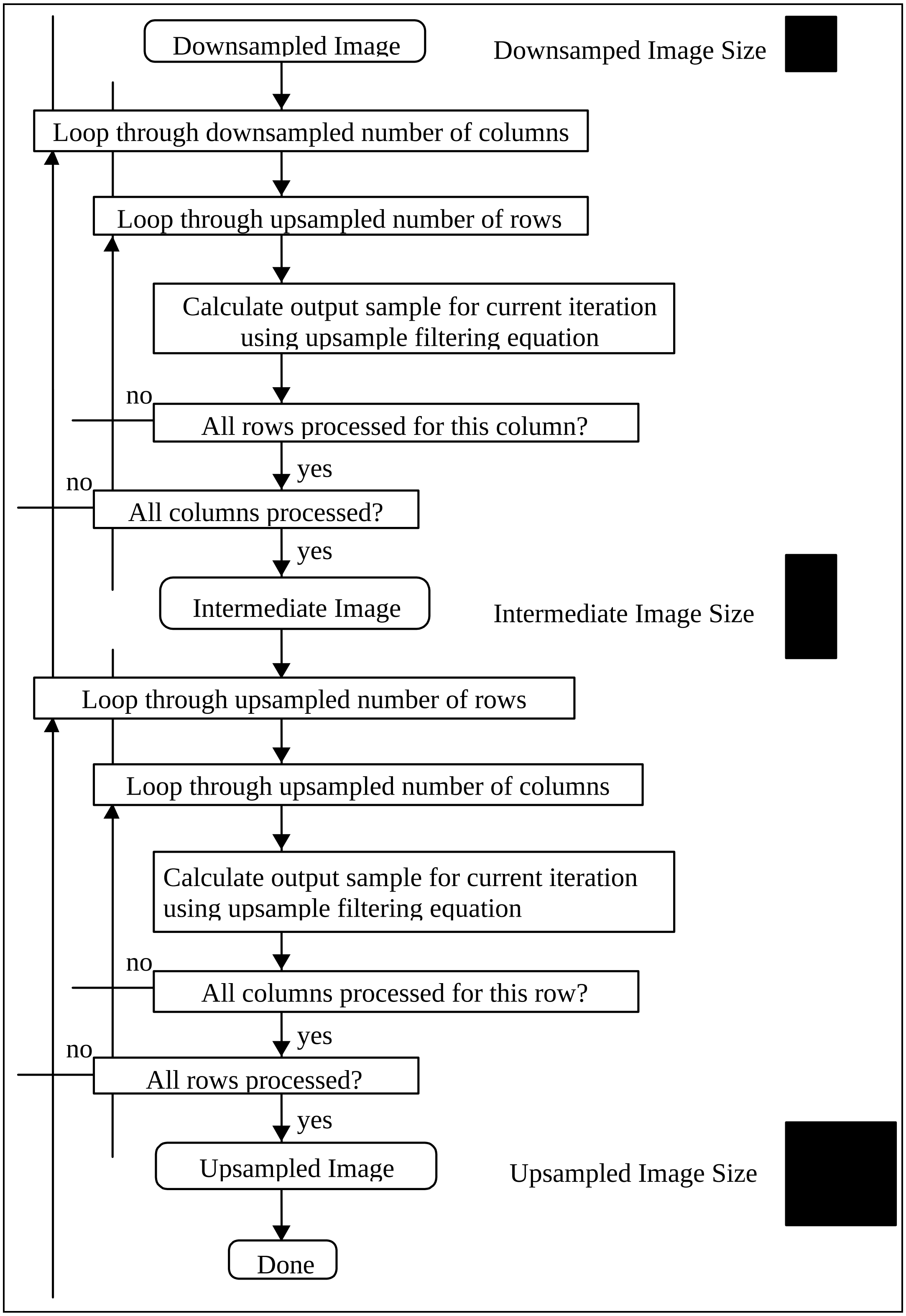

FIGURE 5-10 Control Procedure for Image Upsampling (Row-Column Order)

5.3 Notes

This section is informative only. Comments, explanations, and warnings about the compression system defined in this document are given as ancillary information. The formal requirements are outlined in Sections 5.1. and 5.2.

5.3.1 Image upsampling

Certain applications may require faster upsampling at the cost of much reduced image quality. This is typically the case when computational resources are limited. Bilinear interpolation is often identified as being suitable for these applications since each interpolated value is a function of only the four nearest pixels in two-dimensions, which is a significant decrease in complexity. Furthermore, standard software routines and dedicated hardware exists to perform fast bilinear interpolation. The danger of using a standard package to perform upsampling is that the result will be a shifted version of what is expected. This is due to a phase shift in the downsampled image described by the following equation.

Offset in the downsampled image:

$$O_d = 0.5 - \frac{0.5}{R}$$

where *R* is the upsample ratio discussed in Section 5.2.4.2.1. To ensure proper decoding, the offset must be removed during the upsampling process. The recommended method to reduce the complexity of the decoder while maintaining the proper pixel sampling positions is to reduce the filter length parameter from the value of four to two. It must be emphasized that the quality of the decoded image is noticeably worse than nominal when bilinear interpolation is used. The tradeoff between complexity and quality must be evaluated carefully before deciding to reduce the length of the upsampling filter.

5.3.2 Overlays

Overlays for the decoded images are intolerant to changes in image size. To prevent difficulties with overlays, the upsampled image is constrained to have dimensions equivalent to the original, uncompressed image, and the upsampling algorithm should conform to the specifications outlined in Section 5.2.4. If the decoder does not properly upsample the image, the overlays will be placed at incorrect locations in the image. Such an error could have serious implications for imagery users.

5.3.3 Inherent quality losses

The compression system specified in this document is aimed towards applications requiring very low bit rate compression. The nature of this requirement dictates that much information will be lost in the compression process, albeit minimization of this loss is the objective of the algorithm design. Therefore, visible distortions and resolution degradation are to be expected in the resulting images (e.g. NIIRS losses greater than 1.0 are not uncommon). It should be emphasized that this compressor is not meant for high quality compression of images. The algorithm was designed to provide images of sufficient quality to assist the the decision-making process.

5.3.4 Incompatibility issues with previous generations of NITFS systems

Historical NITFS systems will not have the capability to decode images that have been compressed using the algorithm specified in this document. A new image compression type (IC field in NITF image subheader) has been created to signal the lack of backwards compatibility. Previous generation systems will be unable to decode these images since they will not recognize this new compression type. The IC field value, I1, was for this purpose. Previous systems could conceivably decode the JPEG compressed downsampled image data (see Figure 5-2), but be unable to upsample the data to its proper size. As noted in Section 5.3.2, this could have serious consequences if overlay information is present in the image product.

6.0 ADAPTIVE RECURSIVE INTERPOLATED DIFFERENTIAL PULSE CODE MODULATION (ARIDPCM)COMPRESSION

The requirements to be met by NITFS-compliant systems utilizing the ARIDPCM algorithm are established in MIL-STD-188-197A. The Military Standard provides the technical detail of the NITFS compression algorithm designated by the code C2 in the image compression field of the image subheader, ARIDPCM, for both 8- and 11- bit grey scale imagery. It also provides the required default ARIDPCM Quantization Tables for use in NITFS compliant Secondary Imagery Dissemination Systems (SIDS).

MIL-STD-2500B deletes the requirement to support ARIDPCM compressed imagery, except in the case of archived imagery.

7.0 BI-LEVEL IMAGE COMPRESSION

The requirements to be met by NITFS-compliant systems utilizing the Bi-Level Image Compression algorithm are established in MIL-STD-188-196. The Military Standard establishes the requirements to be met by NITFS systems when image data are compressed using the bi-level facsimile compression specified by the International Telecommunications Union (ITU) International Telegraph and Telephone Consultative Commitee (CCITT) Recommendation T.4 and MIL-STD 188-161C for Group 3 facsimile devices. No attempt has been made to discuss image scanning, communication, or printing systems. The Military Standard provides technical detail for the NITFS compression algorithm designated by the Code C1 in the image compression field of the image subheader for bi-level images or overlays. It also provides the required run-length code tables for use in Secondary Imagery Dissemination Systems (SIDS) in complying with NITFS.

8.0 VECTOR QUANTIZATION COMPRESSION

The requirements to be met by NITFS-compliant systems utilizing the Vector Quantization (VQ) compression algorithm are established in MIL-STD-188-199. This allows the NITFS-compliant systems to accept and decompress data that are compressed using a VQ compression scheme. The Military Standard describes the VQ compression, but does not fully describe the steps for compression. However, the steps involved in decompressing images compressed with VQ are fully described in the Military Standard. The Military Standard provides technical detail of the NITFS VQ decompression algorithm designated by the code C4 or M4 in the image compression field of the image subheader in a NITF file.

ISO 12087, Basic Image Interchange Format (BIIF), defines the VQ decompression algorithm in a normative annex to the standard.

APPENDIX A DEFINITIONS, ACRONYMS AND SYMBOLS

The following definitions are applicable for the purpose of this document. In addition, terms used in this document and defined in the FED-STD-1037B shall use the FED-STD-1037B definition unless noted.

A.1. Acronyms

See ISO/IEC 10918-1 and ISO/IEC 10918-3 for other acronyms used in this document

JIEO	Joint Interoperability and Engineering Organization (formerly JTC^3A)
NITF	National Imagery Transmission Format
NITFS	National Imagery Transmission Format Standard
RGB	Red, Green, Blue
DCT	Discrete Cosine Transform
JPEG	Joint Photographic Experts Group
JBIG	Joint Bi-level Image Experts Group
JFIF	JPEG File Interchange Format
JTIP	JPEG Tiled Image Pyramid
PTSMC	Profiles, Tags, color Spaces, Markers, and Compression type
SPIFF	Still Picture Interchange File Format

A.2. Definitions

The following definitions are applicable for the purpose of this document. In addition, terms used in this profile and defined in the FED-STD-1037B shall use the FED-STD-1037B definition unless noted. These definitions are in addition to the definitions used in ITU-T T.81 I ISO/IEC 10918-1 and ITU-T T.84 | ISO/IEC 0918-3.

PTSMC Authority	Agency or institution charged with managing the registered items.

PTSMC Registration	Official unique listing of a profile, tag, color space, marker, or compression type.
Bit-rate	The average number of bits spent per image sample in a compressed image data file.
Custom Tables	Quantization and Huffman Tables, other than those shown in Appendix A & B of Mil-Std-188-198A. Generating Huffman Tables is described in Appendix C & D of Mil-Std-188-198A.
Default Tables	Quantization and Huffman Tables detailed in Appendix A & B of Mil-Std-188-198A.
Downsampling	A process by which an image’s dimensions, either the number of samples per row or the number of rows or both, are reduced.
Embedded Tables	Quantization and or Huffman Tables that are included with the JPEG entropy encoded data stream within a NITF file.
Non-embedded Tables	Quantization and/or Huffman Tables that are used to generate an entropy encoded data stream but are not included with this encoded data within a NITF file.
Profile	A specific set of capabilities and parameter values or ranges.
Upsampling	A process by which an image’s dimensions, either the number of samples per row or the number of rows or both, are increased
ABPP	This field shall contain the number of “significant bits” for the value in each band of each pixel without compression. Even when the image is compressed, ABPP contains the number of significant bits per pixel that were present in the image before compression. See MIL-STD-2500A, page 38.
NBPP	If IC contains “NC”, “NM”, “C4” or “M4” the field shall contain the number of storage bits used for the value from each component of a pixel vector. If IC = “C3”, this field shall contain the value 8 or the value 12. See MIL-STD-2500A, page 44.

Image Bits	Byte offset 17 of the NITF App6 application data segment. It indicates the original image sample precision. This is a 1 byte field. NITFS allows for any value from 1 - 16, (or 0x01 to 0F). See MIL-STD-188-198A, page 60.
JPEG	Process Byte offset 19 of the NITF App6 application data segment signals which JPEG process follows. This field is set to 0x01 for baseline DCT sequential huffman coding, 8bpp sample precision. Baseline DCT sequential mode is indicated by using the 0xFFC0 frame header in subsequent JPEG frames or blocks. This field is set to 0x04 for extended sequential DCT, Huffman coding, 12bpp sample precision. Extended DCT sequential mode is indicated by using the 0xFFC1 frame header in subsequent JPEG frames or blocks. (Note: From Draft NITF Lossless JPEG documents and ISO 10918-1 Table B.2, Extended Sequential DCT JPEG can also be applied to 8bpp sample precision imagery.) See MIL-STD-188-198A, page 60.
Stream Bits	Byte offset 22 of the NITF App_6 application data segment. This is a 1 byte field. Allowed values are 0x08 and 0x0C, for 8-bit compression and 12-bit JPEG compression. See MIL-STD-188-198A, page 61.

A.3. Symbols

See ISO/IEC 10918-1 and ISO/IEC 10918-3 for definition of symbols used in this profile.

α	Down/upsampling filter length. In the same image space as x_j.
b_i	Beginning summation index. In the same image space as x_j.
ceil ( )	Ceiling function, rounds to next integer toward positive infinity.

e_i	Ending summation index. In the same image space as x_j. floor (·)Floor function, rounds to next integer toward negative infinity.
M_d	Number of samples per row in the downsampled image data.
M_o	Number of samples per row in the original image data.
N_d	Number of rows in the downsampled image data.
N_o	Number of rows in the original image data.
R, R_{row}, R_{col}	Downsampling ratios. Equal to the ratio of the number of samples in a given dimension of the original image to that of the same dimension in the downsampled image.
round ( )	Round function, rounds toward nearest integer.
w_{ij}	Downsampling or upsampling filter coefficient relating x_j to y_i.
x_j	Input image sample in the downsampling or upsampling formulae.
y_i	Output image sample in the downsampling or upsampling formulae.

APPENDIX B: COMPRESSION PARAMETERS FOR DOWNSAMPLED JPEG

B.1 Compression parameters

The recommended compression parameters allows images to be coded at five different quality levels, which will be referred to as IQ1, IQ2, IQ3, IQ4, and IQ5. IQ5 (quality level 5) compression has the highest fidelity to the source image, but achieves the least compression. IQ1 compression results in the worst reconstructed image quality, but the highest compression. The IQ2, IQ3, and IQ4 levels represent compromises between IQ1 and IQ5 in ascending order of quality, and descending order of compression.

In the following sections, the compression parameters are given for both 8-bit and 12-bit source images. The pertinent parameters for each quality level are the downsample ratio (Refer to Section 3.2.1), JPEG quantization table (Refer to MIL-STD-188-198A), and the JPEG Huffman tables (Refer to MIL-STD-188-198A). Please note that these parameters are currently not fully optimized. Additionally, the parameters were developed only for the 8-bit and 12-bit classes of visible imagery. Parameters which are fully optimized across a wide range of image classes may become available in a future version of this document.

B.2 Eight-bit gray scale compression parameters

B.2.1 Downsample ratios

The downsample ratios for each quality level is shown in the following table. The downsample ratio at a particular quality level is applied as specified in Section 3.2.1.

TABLE B-1 Downsample ratios for 8-bit gray scale images

Quality Level	Downsample Ratio
IQ1	27.0
IQ2	13.5
IQ3	7.5
IQ4	4.5
IQ5	1.5

B.2.2 JPEG Quantization Table

The following quantization table applies to all quality levels (IQ1-IQ5). The values are formatted as an 8x8 matrix of quantization scale factors. The format is such that the matrix entry at a particular index location is the quantization factor to be applied to the DCT frequency coefficient at the matching index location. For example, the first element, located in the top-left position, is used to quantize the DC DCT frequency coefficient.

TABLE B-2 8-bit gray scale JPEG quantization table

36	36	37	39	42	45	50	54
36	37	39	42	45	50	54	60
37	39	42	45	50	54	60	66
39	42	45	50	54	60	66	74
42	45	50	54	60	66	74	81
45	50	54	60	66	74	81	90
50	54	60	66	74	81	90	99
54	60	66	74	81	90	99	110

B.2.3 JPEG Huffman Tables

The parameters defined in Sections B.2.3.1 through B.2.3.5 specify the DC and AC BITS and HUFFVAL tables needed to generate Huffman codes for the pre-defined quality levels. BITS and HUFFVAL are described in the JPEG standard document, MIL-STD-188-198A. The "luminance" label is to emphasize that these tables are meant for the coding of gray scale images.

B.2.3.1 IQ1 Huffman Table parameters

dc_luminance_bits[16] = 0, 3, 1, 1, 1, 1, 0, 2, 3, 0, 0, 0, 0, 0, 0, 0
dc_luminance_val[12] = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
ac_luminance_bits[16] = 0, 2, 2, 1, 3, 3, 2, 4, 6, 1, 1, 1, 1, 0, 0,135
ac_luminance_val[162] = 0, 1, 2, 17, 33, 3, 49, 65, 18, 81,
97, 113, 129, 19, 34, 145, 161, 4, 50, 177,
193, 209, 240, 66, 225, 82, 241, 20, 35, 51,
98, 5, 114, 130, 146, 67, 83, 162, 21, 52,
210, 36, 37, 178, 6, 7, 8, 9, 10, 22,
23, 24, 25, 26, 38, 39, 40, 41, 42, 53,
54, 55, 56, 57, 58, 68, 69, 70, 71, 72,
73, 74, 84, 85, 86, 87, 88, 89, 90, 99,
100, 101, 102, 103, 104, 105, 106, 115, 116, 117,
118, 119, 120, 121, 122, 131, 132, 133, 134, 135,
136, 137, 138, 147, 148, 149, 150, 151, 152, 153,
154, 163, 164, 165, 166, 167, 168, 169, 170, 179,
180, 181, 182, 183, 184, 185, 186, 194, 195, 196,
197, 198, 199, 200, 201, 202, 211, 212, 213, 214,
215, 216, 217, 218, 226, 227, 228, 229, 230, 231,
232, 233, 234, 242, 243, 244, 245, 246, 247, 248,
249, 250

B.2.3.2 IQ2 Huffman Table parameters

dc_luminance_bits[16] = 0, 3, 1, 1, 1, 1, 0, 2, 3, 0, 0, 0, 0, 0, 0, 0
dc_luminance_val[12] = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
ac_luminance_bits[16] = 0, 1, 3, 3, 3, 2, 3, 8, 2, 1, 1, 1, 1, 0, 2, 131
ac_luminance_val[162] = 1, 0, 2, 17, 3, 33, 49, 18, 65, 81,
97, 113, 34, 129, 145, 4, 19, 50, 161, 177,
193, 209, 240, 66, 225, 82, 241, 20, 35, 98,
5, 51, 114, 146, 67, 130, 83, 162, 21, 36,
52, 178, 99, 115, 147, 194, 210, 6, 7, 8,
9, 10, 22, 23, 24, 25, 26, 37, 38, 39,
40, 41, 42, 53, 54, 55, 56, 57, 58, 68,
69, 70, 71, 72, 73, 74, 84, 85, 86, 87,
88, 89, 90, 100, 101, 102, 103, 104, 105, 106,
116, 117, 118, 119, 120, 121, 122, 131, 132, 133,
134, 135, 136, 137, 138, 148, 149, 150, 151, 152,
153, 154, 163, 164, 165, 166, 167, 168, 169, 170,
179, 180, 181, 182, 183, 184, 185, 186, 195, 196,
197, 198, 199, 200, 201, 202, 211, 212, 213, 214,
215, 216, 217, 218, 226, 227, 228, 229, 230, 231,
232, 233, 234, 242, 243, 244, 245, 246, 247, 248,
249, 250

B.2.3.3 IQ3 Huffman Table parameters.

dc_luminance_bits[16] = 0, 3, 1, 1, 1, 1, 0, 2, 3, 0, 0, 0, 0, 0, 0, 0
dc_luminance_val[12] = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
ac_luminance_bits[16] = 0, 1, 3, 3, 3, 2, 3, 8, 2, 1, 1, 1, 1, 0, 2, 131
ac_luminance_val[162] = 1, 0, 2, 17, 3, 33, 49, 18, 65, 81,
97, 113, 34, 129, 145, 4, 19, 50, 161, 177,
193, 209, 240, 66, 225, 82, 241, 35, 20, 51,
98, 114, 130, 146, 5, 83, 67, 162, 21, 99,
36, 52, 178, 210, 6, 7, 8, 9, 10, 22,
23, 24, 25, 26, 37, 38, 39, 40, 41, 42,
53, 54, 55, 56, 57, 58, 68, 69, 70, 71,
72, 73, 74, 84, 85, 86, 87, 88, 89, 90,
100, 101, 102, 103, 104, 105, 106, 115, 116, 117,
118, 119, 120, 121, 122, 131, 132, 133, 134, 135,
136, 137, 138, 147, 148, 149, 150, 151, 152, 153,
154, 163, 164, 165, 166, 167, 168, 169, 170, 179,
180, 181, 182, 183, 184, 185, 186, 194, 195, 196,
197, 198, 199, 200, 201, 202, 211, 212, 213, 214,
215, 216, 217, 218, 226, 227, 228, 229, 230, 231,
232, 233, 234, 242, 243, 244, 245, 246, 247, 248,
249, 250

B.2.3.4 IQ4 Huffman Table parameters

dc_luminance_bits[16] =0, 3, 1, 1, 1, 1, 0, 2, 3, 0, 0, 0, 0, 0, 0, 0
dc_luminance_val[12] = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
ac_luminance_bits[16] =0, 1, 3, 3, 3, 2, 3, 8, 2, 1, 1, 1, 1, 0, 2, 131
ac_luminance_val[162] = 1, 0, 2, 17, 3, 33, 49, 18, 65, 81,
97, 113, 34, 129, 145, 4, 19, 50, 161, 177,
193, 209, 240, 66, 225, 82, 241, 35, 20, 51,
98, 114, 146, 5, 67, 130, 162, 83, 178, 36,
99, 21, 52, 115, 210, 6, 7, 8, 9, 10,
22, 23, 24, 25, 26, 37, 38, 39, 40, 41,
42, 53, 54, 55, 56, 57, 58, 68, 69, 70,
71, 72, 73, 74, 84, 85, 86, 87, 88, 89,
90, 100, 101, 102, 103, 104, 105, 106, 116, 117,
118, 119, 120, 121, 122, 131, 132, 133, 134, 135,
136, 137, 138, 147, 148, 149, 150, 151, 152, 153,
154, 163, 164, 165, 166, 167, 168, 169, 170, 179,
180, 181, 182, 183, 184, 185, 186, 194, 195, 196,
197, 198, 199, 200, 201, 202, 211, 212, 213, 214,
215, 216, 217, 218, 226, 227, 228, 229, 230, 231,
232, 233, 234, 242, 243, 244, 245, 246, 247, 248,
249, 250

B.2.3.5 IQ5 Huffman Table parameters

dc_luminance_bits[16] =0, 3, 1, 1, 1, 1, 0, 2, 3, 0, 0, 0, 0, 0, 0, 0
dc_luminance_val[12] = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
ac_luminance_bits[16] =0, 1, 3, 2, 5, 2, 3, 7, 4, 1, 1, 1, 1, 1, 1, 129
ac_luminance_val[162] = 1, 0, 2, 17, 33, 49, 3, 18, 65, 81,
97, 113, 129, 34, 145, 161, 4, 19, 50, 177,
193, 209, 240, 66, 82, 225, 241, 35, 20, 51,
98, 114, 130, 146, 5, 162, 21, 52, 67, 83,
178, 6, 7, 8, 9, 10, 22, 23, 24, 25,
26, 36, 37, 38, 39, 40, 41, 42, 53, 54,
55, 56, 57, 58, 68, 69, 70, 71, 72, 73,
74, 84, 85, 86, 87, 88, 89, 90, 99, 100,
101, 102, 103, 104, 105, 106, 115, 116, 117, 118,
119, 120, 121, 122, 131, 132, 133, 134, 135, 136,
137, 138, 147, 148, 149, 150, 151, 152, 153, 154,
163, 164, 165, 166, 167, 168, 169, 170, 179, 180,
181, 182, 183, 184, 185, 186, 194, 195, 196, 197,
198, 199, 200, 201, 202, 210, 211, 212, 213, 214,
215, 216, 217, 218, 226, 227, 228, 229, 230, 231,
232, 233, 234, 242, 243, 244, 245, 246, 247, 248,
249, 250

B.3 Twelve-bit gray scale compression parameters

B.3.1 Downsample ratios

The downsample ratios for each quality level is shown in the following table. The downsample ratio at a particular quality level is applied as specified in Section 3.2.1. Note that the downsample ratios for 12-bit images are the same as for 8-bit images.

TABLE B-3 Downsample ratios for 12-bit gray scale images

Quality Level	Downsample Ratio
IQ1	27.0
IQ2	13.5
IQ3	7.5
IQ4	4.5
IQ5	1.5

B.3.2 JPEG Quantization Table

The following quantization table applies to all quality levels (IQ1-IQ5). The format of the table shown below follows the format used to describe the 8t-bit gray scale JPEG Quantization Table in Section B.2.2.

TABLE B-4 Twelve-bit gray scale JPEG Quantization Table

576	576	592	624	672	720	800	864
576	592	624	672	720	800	864	960
592	624	672	720	800	864	960	1056
624	672	720	800	864	960	1056	1184
672	720	800	864	960	1056	1184	1296
720	800	864	960	1056	1184	1296	1440
800	864	960	1056	1184	1296	1440	1584
864	960	1056	1184	1296	1440	1584	1760

B.3.3 JPEG Huffman Tables

The parameters defined in Sections B.3.3.1 through B.3.3.5 specify the DC and AC BITS and HUFFVAL tables needed to generate Huffman codes for the pre-defined quality levels. These tables apply to compression of 12-bit imagery only. An explanation of the format used to specify the Huffman parameters can be found in Section B.2.3.

B.3.3.1 IQ1 Huffman Table generation parameters

dc_luminance_bits[16] = 0, 3, 1, 1, 1, 1, 0, 0, 6, 3, 0, 0, 0, 0, 0, 0
dc_luminance_val[16] = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
ac_luminance_bits[16] =0, 2, 2, 1, 3, 3, 2, 4, 6, 0, 1, 1, 0, 2, 0, 199
ac_luminance_val[226] = 0, 1, 2, 17, 33, 3, 49, 65, 18, 81,
97, 113, 129, 4, 19, 34, 145, 50, 161, 177,
193, 209, 240, 66, 225, 82, 241, 20, 35, 51,
98, 114, 130, 146, 5, 67, 83, 162, 21, 36,
178, 210, 99, 6, 7, 8, 9, 10, 11, 12,
13, 14, 22, 23, 24, 25, 26, 27, 28, 29,
30, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 68, 69, 70, 71, 72, 73, 74, 75,
76, 77, 78, 84, 85, 86, 87, 88, 89, 90,
91, 92, 93, 94, 100, 101, 102, 103, 104, 105,
106, 107, 108, 109, 110, 115, 116, 117, 118, 119,
120, 121, 122, 123, 124, 125, 126, 131, 132, 133,
134, 135, 136, 137, 138, 139, 140, 141, 142, 147,
148, 149, 150, 151, 152, 153, 154, 155, 156, 157,
158, 163, 164, 165, 166, 167, 168, 169, 170, 171,
172, 173, 174, 179, 180, 181, 182, 183, 184, 185,
186, 187, 188, 189, 190, 194, 195, 196, 197, 198,
199, 200, 201, 202, 203, 204, 205, 206, 211, 212,
213, 214, 215, 216, 217, 218, 219, 220, 221, 222,
226, 227, 228, 229, 230, 231, 232, 233, 234, 235,
236, 237, 238, 242, 243, 244, 245, 246, 247, 248,
249, 250, 251, 252, 253, 254

B.3.3.2 IQ2 Huffman Table generation parameters

dc_luminance_bits[16] =0, 3, 1, 1, 1, 1, 0, 0, 6, 3, 0, 0, 0, 0, 0, 0
dc_luminance_val[16] = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
ac_luminance_bits[16] =0, 1, 3, 3, 3, 2, 3, 8, 2, 0, 1, 1, 1, 0, 1, 197
ac_luminance_val[226] = 1, 0, 2, 17, 3, 33, 49, 18, 65, 81,
97, 113, 34, 129, 145, 4, 19, 50, 161, 177,
193, 209, 240, 66, 225, 82, 241, 35, 98, 20,
114, 51, 130, 146, 67, 5, 83, 162, 178, 210,
21, 194, 6, 7, 8, 9, 10, 11, 12, 13,
14, 22, 23, 24, 25, 26, 27, 28, 29, 30,
36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 68, 69, 70, 71, 72, 73, 74, 75,
76, 77, 78, 84, 85, 86, 87, 88, 89, 90,
91, 92, 93, 94, 99, 100, 101, 102, 103, 104,
105, 106, 107, 108, 109, 110, 115, 116, 117, 118,
119, 120, 121, 122, 123, 124, 125, 126, 131, 132,
133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 163, 164, 165, 166, 167, 168, 169, 170,
171, 172, 173, 174, 179, 180, 181, 182, 183, 184,
185, 186, 187, 188, 189, 190, 195, 196, 197, 198,
199, 200, 201, 202, 203, 204, 205, 206, 211, 212,
213, 214, 215, 216, 217, 218, 219, 220, 221, 222,
226, 227, 228, 229, 230, 231, 232, 233, 234, 235,
236, 237, 238, 242, 243, 244, 245, 246, 247, 248,
249, 250, 251, 252, 253, 254

B.3.3.3 IQ3 Huffman Table generation parameters

dc_luminance_bits[16] =0, 3, 1, 1, 1, 0, 0, 5, 5, 0, 0, 0, 0, 0, 0, 0
dc_luminance_val[16] = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15
ac_luminance_bits[16] =0, 1, 3, 3, 3, 2, 3, 8, 2, 0, 1, 1, 1, 0, 1, 197
ac_luminance_val[226] = 1, 0, 2, 17, 3, 33, 49, 18, 65, 81,
97, 113, 34, 129, 145, 4, 19, 50, 161, 177,
193, 209, 240, 66, 225, 241, 35, 82, 98, 20,
114, 5, 21, 51, 130, 162, 226, 99, 146, 67,
83, 36, 178, 194, 210, 6, 7, 8, 9, 10,
11, 12, 13, 14, 22, 23, 24, 25, 26, 27,
28, 29, 30, 37, 38, 39, 40, 41, 42, 43,
44, 45, 46, 52, 53, 54, 55, 56, 57, 58,
59, 60, 61, 62, 68, 69, 70, 71, 72, 73,
74, 75, 76, 77, 78, 84, 85, 86, 87, 88,
89, 90, 91, 92, 93, 94, 100, 101, 102, 103,
104, 105, 106, 107, 108, 109, 110, 115, 116, 117,
118, 119, 120, 121, 122, 123, 124, 125, 126, 131,
132, 133, 134, 135, 136, 137, 138, 139, 140, 141,
142, 147, 148, 149, 150, 151, 152, 153, 154, 155,
156, 157, 158, 163, 164, 165, 166, 167, 168, 169,
170, 171, 172, 173, 174, 179, 180, 181, 182, 183,
184, 185, 186, 187, 188, 189, 190, 195, 196, 197,
198, 199, 200, 201, 202, 203, 204, 205, 206, 211,
212, 213, 214, 215, 216, 217, 218, 219, 220, 221,
222, 227, 228, 229, 230, 231, 232, 233, 234, 235,
236, 237, 238, 242, 243, 244, 245, 246, 247, 248,
249, 250, 251, 252, 253, 254

B.3.3.4 IQ4 Huffman Table generation parameters

dc_luminance_bits[16] =0, 3, 1, 1, 1, 0, 0, 5, 5, 0, 0, 0, 0, 0, 0, 0
dc_luminance_val[16] = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15
ac_luminance_bits[16] =0, 1, 3, 3, 3, 2, 3, 8, 1, 2, 1, 1, 1, 0, 2, 195
ac_luminance_val[226] = 1, 0, 2, 17, 3, 33, 49, 18, 65, 81,
97, 113, 34, 129, 145, 4, 19, 50, 161, 177,
193, 209, 240, 225, 66, 82, 241, 35, 98, 20,
51, 114, 130, 162, 226, 5, 21, 99, 146, 67,
178, 36, 83, 115, 194, 210, 6, 7, 8, 9,
10, 11, 12, 13, 14, 22, 23, 24, 25, 26,
27, 28, 29, 30, 37, 38, 39, 40, 41, 42,
43, 44, 45, 46, 52, 53, 54, 55, 56, 57,
58, 59, 60, 61, 62, 68, 69, 70, 71, 72,
73, 74, 75, 76, 77, 78, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 100, 101, 102,
103, 104, 105, 106, 107, 108, 109, 110, 116, 117,
118, 119, 120, 121, 122, 123, 124, 125, 126, 131,
132, 133, 134, 135, 136, 137, 138, 139, 140, 141,
142, 147, 148, 149, 150, 151, 152, 153, 154, 155,
156, 157, 158, 163, 164, 165, 166, 167, 168, 169,
170, 171, 172, 173, 174, 179, 180, 181, 182, 183,
184, 185, 186, 187, 188, 189, 190, 195, 196, 197,
198, 199, 200, 201, 202, 203, 204, 205, 206, 211,
212, 213, 214, 215, 216, 217, 218, 219, 220, 221,
222, 227, 228, 229, 230, 231, 232, 233, 234, 235,
236, 237, 238, 242, 243, 244, 245, 246, 247, 248,
249, 250, 251, 252, 253, 254

B.3.3.5 IQ5 Huffman Table generation parameters

dc_luminance_bits[16] =0, 3, 1, 1, 1, 1, 0, 0, 6, 3, 0, 0, 0, 0, 0, 0
dc_luminance_val[16] = 0, 2, 3, 1, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15
ac_luminance_bits[16] =0, 1, 3, 2, 5, 2, 3, 8, 1, 2, 1, 1, 1, 1, 0, 195
ac_luminance_val[226]= 1, 0, 2, 17, 33, 49, 3, 18, 65, 81,
97, 113, 129, 34, 145, 161, 4, 19, 50, 177,
193, 209, 225, 240, 66, 35, 82, 98, 241, 20,
51, 114, 130, 146, 67, 162, 194, 178, 210, 5,
6, 7, 8, 9, 10, 11, 12, 13, 14, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 36,
37, 38, 39, 40, 41, 42, 43, 44, 45, 46,
52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
62, 68, 69, 70, 71, 72, 73, 74, 75, 76,
77, 78, 83, 84, 85, 86, 87, 88, 89, 90,
91, 92, 93, 94, 99, 100, 101, 102, 103, 104,
105, 106, 107, 108, 109, 110, 115, 116, 117, 118,
119, 120, 121, 122, 123, 124, 125, 126, 131, 132,
133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 163, 164, 165, 166, 167, 168, 169, 170,
171, 172, 173, 174, 179, 180, 181, 182, 183, 184,
185, 186, 187, 188, 189, 190, 195, 196, 197, 198,
199, 200, 201, 202, 203, 204, 205, 206, 211, 212,
213, 214, 215, 216, 217, 218, 219, 220, 221, 222,
226, 227, 228, 229, 230, 231, 232, 233, 234, 235,
236, 237, 238, 242, 243, 244, 245, 246, 247, 248,
249, 250, 251, 252, 253, 254

APPENDIX C PIXEL DEPTH SCENARIO TABLE (GUIDELINES FOR SCALING)

C.1 General

This appendix is provided as a guide to populating various fields of the image subheader and JPEG header under various forms of image compression.

C.2. General Requirement

C.2.1 Scenario Table

Table C-1 lists various scenarios for describing compressed image data in a NITF file. Each scenario is detailed in subsequent paragraghs, the table shows what values should appear in the ABPP, and NBPP fields of the image subheader and Image Bits, JPEG Process and Steam Bits fields of the JPEG header.

Table C-1 Pixel Depth Scenario Table

Scenario	Original image data bpp	ABPP	NBPP	Image Bits	JPEG Process	Stream Bits
1	8	8	8	8	1 - baseline	8
2	12	12	12	12	4 - extended	12
3	8	8	8	8	4	12
4	4 thru 7bpp Scaled to 8	8	8	8	1	8
5	9 thru 11	12	12	12	4	12
6	4 thru 7 bpp not scaled. i.e. 5	5	8	5	1	8
7	9 thru 11 not scaled. i.e. 10	10	12	10	4	12
8	>12bpp not scaled					

Table C-1 Pixel Depth Scenario Table - continued

Scenario	Original image data bpp	ABPP	NBPP	Image Bits	JPEG Process	Stream Bits
9	>12bpp scaled to 12bpp i.e. 15	12	12	12	4	12
10	X	X	X	X	14 -Lossless	NA
11	X IOMAPA 12	X	*12*	X	4	12

C.2.1.1 Scenario 1. Simple 8 Bit JPEG.

This scenario represents ordinary 8 bit/pixel image data run through an 8-bit lossy JPEG encoding.

C.2.1.2 Scenario 2. Simple 12 Bit JPEG.

This scenario represents ordinary 12- bit/pixel image data run through a 12-bit lossy JPEG encoding

C.2.1.3 Scenario 3. 8bpp Image Through 12-Bit JPEG.

This is a case in which 8bpp image data is compressed through a 12-bit lossy JPEG encoding algorithm. The implementor would have to feed the algorithm 8bits of image data in a 12-bit structure, the upper 4 bits would, of course, be set to zero.

C.2.1.4 Scenario 4. 4 Thru 7 Bit Scaled To 8.

This is the situation in which image data from 4bpp through 7bpp is JPEG compressed. Before compression the sender or creator of the JPEG stream decided to convert, scale, shift or translate the original image data to full 8bpp then JPEG encode it. In this scenario the receiver of such a file has no way of knowing what kind of conversion took place before JPEG compression.

C.2.1.5 Scenario 5. 9 Thru 11 Scaled To 12.

Almost identical to scenario #4. This is the situation in which image data from 9bpp through 11bpp is JPEG compressed through the 12-bit lossy algorithm. Before compression the sender or creator of the JPEG stream decided to convert, scale, shift or translate the original image data to full 12bpp then JPEG encode it.

C.2.1.6 Scenario 6. 4 Thru 7 Not Scaled

This is the situation in which a low bit depth device is generating imagery. The data is not scaled or altered before 8-bit JPEG compression. This implies that higher order bits are set to zero.

C.2.1.7 Scenario 7. 9 Thru 11 Not Scaled

This is the situation in which a higher bit depth device is generating imagery. The data is not scaled or shifted before 12-bit JPEG compression. Again this implies that higher order bits are set to zero.

C.2.1.8 Scenario 8. Greater Than 12bit Data Not Scaled

This situation is not allowed in the current version of Lossy NITF JPEG. Image data greater than 12bpp can not be JPEG compressed by either of the NITF adopted sequential DCT encoding schemes.

C.2.1.9 Scenario 9. Greater Than 12-bit Data Scaled

This scenario represents an implementation accepting a high bit depth image data from a collector, then scaling it down to 12bpp of significant data then JPEG encoding it Since it is now 12-bit JPEG compressed any NITF system will be able to handle it.

C.2.1.10 Scenario 10 X bpp Lossless Compression

This is the situation in which image data of any pixel depth, 2 through 16 bpp, is compressed by a lossless JPEG implementation.

C.2.1.11 Scenario 11.X In 12 IOMAPA

This scenario represents situations in which pre processing is applied to an original image of any pixel depth from 4- to 12-bits, translating it to 12bpp image data This transformed data is then JPEG compressed through the Extended DCT routine. The appropriate IOMAPA tag is then included in the image subheader of the NITF file. This allows a receiver to restore, or post process the image to something close to the original X bits.

APPENDIX D: NITF / JPEG INPUT / OUTPUT MAPPING COMPRESSION ALGORITHM

D.1 Note

Although this algorithm is used by some systems, it is not a required algorithm for JPEG Lossy mapping compression.

D.2 Input Amplitude Mapping Function

Prior to input into the JPEG Lossy DCT compression algorithm, the input data is mapped via an amplitude mapping process, when the database value MAP_SELECT is set to a value other than zero. This process is defined as follows:

1. For each image block the minimum pixel intensity is determined. This minimum pixel intensity is included in the NITF/JPEG overhead data.

2. The image block’s minimum pixel intensity is then subtracted from each pixel of the corresponding image block.

3. The resultant pixel intensity is clamped to be greater than or equal to 0 and less than or equal to 4095.

4. The resultant pixel intensity is then mapped via the amplitude mapping function.

D.3 Amplitude Mapping Function Methods

If the MAP_SELECT flag is set to 1, each pixel value, subsequent to subtraction of the image block minimum value is scaled by multiplying each pixel value by the scaling factor 2^S1, where the exponent S1 is a database value. Each resulting pixel is passed through an Input Amplitude Mapping Table. The Input Amplitude Mapping Table and the corresponding Output Amplitude Mapping Table are database items. The Output Amplitude Mapping Table is required for the expansion process and is included in the NITF/JPEG overhead data.

If the MAP_SELECT flag is set to 2, a generalized log mapping is utilized as the basis for the input amplitude mapping function. The parameters R and S1 are utilized to generate this function. These parameters are database items and are included in the NITFS/JPEG overhead data. The input amplitude mapping process is defined as:

IX	=	Original pixel
MIN	=	Minimum pixel of the image block (these values are included in the NITF/JPEG format)
IXX	=	IX-MIN
IMAX	=	4096 for 12bpp JPEG option
IXMAX=		IMAX-1 Maximum input
ISF	=	2^S1 Scale factor where S1=database parameter
ISMAX=		(IMAX / ISF)-1 Scaled Maximum
IXMID=		(IMAX / (2*ISF)) Scaled mid point
A	=	(R-1.) / IXMID where R = database parameter
B	=	IXMAX / Ln(1.+A*ISMAX)

If R not equal to 1:

IY = int [B * Ln (1. + A * IXX) + 0.5] Mapped Output

Where Ln denotes the natural log function, and int () denotes integer truncation.

If R=1:

IY = IXX * ISF Mapped Output

Note: An input mapping table can be generated from the above function so that a table look-up process can be employed for ease of implementation

The output data from this process is checked to ensure that it is greater than or equal to 0 and less than or equal to IXMAX.

If the MAP_SELECT flag is set to three, a segmented polynomial mapping is utilized. The mapping function is defined as three segments, with each segment being defined by a fifth order polynomial. The starting point of each segment and a set of six coefficients defining that input mapping polynomial for each segment are database items. A reciprocal set of coefficients required for the output mapping function are also included as database items. The reciprocal set of coefficients and the breakpoints are included in the NITFS/JPEG overhead data.

Prior to the application of the input mapping polynomial, the data is scaled as follows:

IX = Original pixel

MIN = Minimum pixel of the image block (these values are included in the NITF/JPEG format)

IXX = IX-MIN

IMAX = 4096 for 12bpp JPEG option

IXMAX= IMAX-1 Maximum input

ISF = 2^S1 Scale factor where S1=database parameter

IXXX= IXX*ISF

The resultant IXXX is input into the polynomial segment for which the scaled input IXXX falls withing the segment boundaries as defined by:

Segment (J) is defined as X(J-1) <= IXXX < X(J) J = 1,2,3

where X(I) are the segment boundaries, and X(0) = 0 , X(3) =4096

and X(1) and X(2) are database items defining segment boundaries.

For each segment the polynomial is defined as:

IY= int [(a0 + a1 * IZ + a2*(IZ)^2 + a3*(IZ)^3+ a4*(IZ)^4 + a5*(IZ)^5+0.5)]

where IZ = IXXX - X(J-1)

X(J-1) = Segment (J) Lower boundary

The coefficients a(i) are database items.

After the application of the polynomial mapping function, the data is checked to ensure that it is greater than or equal to 0 and less than or equal to IXMAX.

D.4 Output Amplitude Mapping Function

An output mapping is performed as indicated by the MAP_SELECT flag in the NITF/JPEG format. All data required for the output amplitude mapping process is contained in the NITF/JPEG overhead data.

Prior to the application of the amplitude mapping function , the output of the DCT expansion is clamped to ensure that it is greater than or equal to 0 and less than or equal to 4095.

For each of the amplitude mapping methods, the expander generates a resultant output amplitude mapping table from the data items included in the NITF file, and apply the table to the data output from the JPEG Expansion process.

D.5 Output Amplitude Mapping Methods

If the MAP_SELECT flag is set to zero, no output mapping function is performed. However, if the S2 field is not equal to zero, the data value is scaled by the factor of 2^S2. The output scaled pixel value uses the following expression:

OX = int [(IY / OSF)]

where:

IY = pixel value from JPEG expander

OSF = 2^S2

OX = Output precision scaled pixel value

If the MAP_SELECT flag is set to one, the output mapping is performed using the table included in the NITF/JPEG overhead data. The mapped data output is then scaled by dividing each output pixel value by the scale factor (2 ^ (S1+S2)) where the scale factor exponents S1 and S2 are included in the NITF/JPEG overhead data.

If the MAP_SELECT flag is set to the value 2, the following generalized log mapping is utilized for each pixel output from the JPEG expansion process.

If R is not equal to 1.0

IX = int (((exp (IY/B) -1.)A) / OSF + 0.5)

If R = 1.0

IX = int ((IY / ISF * OSF) + 0.5)

where:

IX = output mapped pixel before MIN_SCALE value added (also includes rescaling is S2 > 0).

R = a log ratio which is included in the NITF/JPEG overhead data

IY = the output pixel from the JPEG expansion process

A = (R-1.) / IXMID

B = IXMAX / Ln(1. + A * ISMAX)

ISMAX= (IMAX / ISF) -1

IXMID = (IMAX / (2*ISF))

IMAX = 4096 for 12bpp JPEG

IXMAX= IMAX -1

ISF = 2^S1

S1 = a scale factor exponent which is included in the NITF/JPEG overhead data

Ln() = denotes natural log function

int() = denotes integer truncation

OSF = 2^S2

S2 = a scale factor exponent which is included in the NITF/JPEG overhead data

If the MAP_SELECT flag is set to the value 3, the following segmented polynomial mapping is utilized for each pixel output from the JPEG expansion process.

The output pixel (IY) from the JPEG expansion process determines which segment of the polynomial function is utilized

Segment (J) is defined as

X(J-1) <= IY < X(J) for J=1,2,3

where:

X(J) are segment bounds

X(0) =0, X(3)=4096

X(1) and X(2) are included in the NITF/JPEG overhead data.

The output pixel value (IY) is mapped as follows using the coefficients (bi) for the appropriate polynomial segment as defined above.

$$IXX=int[(b0 + b1*IZ + b2*(IZ)^2 + b3*(IZ)^3 + b4*(IZ)^4 + b5*(IZ)^5)+0.5]$$

$$IZ=IY - X(J)$$

The coefficients bi are included in the NITF/JPEG format and where X(J) is segment (J)'s lower boundary

The output of the polynomial mapping function (IXX) is scaled by the following relationship:

IX = int ((IXX / ISF) +0.5)

where:

ISF = 2^(S1 + S2)

S1 = a scale factor exponent which is included in the NITF/JPEG overhead data

S2 = a scale factor exponent which is included in the NITF/JPEG overhead data

IX = rescaled output mapped pixel

int = denotes integer truncation

After performing the output mapping function specified above, when the MAP_SELECT flag is set to any value other than zero, the scaled minimum pixel value for each particular image block is added to the output amplitude mapped pixel to yield the resultant output pixel.

The scaled minimum pixel value (MIN_SCALE) is defined as

MIN_SCALE = int ((min / 2^S2) + 0.5)

where:

min = the minimum pixel value for the given image block as stored in APP6 / (Extension NITF0001)

S2 = output scale factor exponent

int = denote integer truncation

Both the minimum value (min) and the exponent S2 are included in NITF/JPEG overhead data.

If no output mapping is specified, the minimum pixel value will not be added to the output of the JPEG expansion.

Note: The transmitted output mapping function applies to the entire image.

After the application of the output mapping function and the addition of the minimum value the data is clamped to ensure that is greater than or equal to zero and less than or equal to the value OMAX, where

OMAX = [4096 / 2^ (S1+S2)]-1 where S1 and S2 are scale factor exponents

APPENDIX E HUFFMAN AND QUANTIZATION TABLES

E.1 IR 12-bit tables for the q1 level

16,	16,	84,	572,	954,	1758,	2826,	4096
16,	86,	560,	1008,	1616,	1878,	3150,	4096
84,	562,	1048,	1678,	1914,	2032,	4096,	4096
576,	1014,	1688,	1954,	1612,	4096,	4096,	4096
972,	1638,	1956,	1686,	4096,	4096,	4096,	4096
1722,	1858,	2184,	4096,	4096,	4096,	4096,	4096
2826,	3144,	4096,	4096,	4096,	4096,	4096,	4096
4096,	4096,	4096,	4096,	4096,	4096,	4096,	4096

```
static const UINT8   dc_luminance_bits[17] =

{   0,  0,  2,  2,  3,  1,  1,  1,  1,
  1,  0,  3,  1,  0,  0,  0,  0};


static const UINT8   dc_luminance_val[] =

{   6,   7,   5,   8,   3,   4,   9,   2,  10,   1,
   0,  11,  12,  13,  14,  15,};


static const UINT8   ac_luminance_bits[17] =

{   0,  0,  0,  6,  2,  2,  2,  2,  1,
  4,  0,  1,  1,  0,  0,  2, 203,};
```

```
static const UINT8   ac_luminance_val[] =
{     0,      1,      2,      3,      4,      5,      6,     17,      7,     33,
      8,     18,     19,     49,     65,      9,     20,     34,     81,     97,
     21,     50,    113,     22,     35,    129,    145,     10,     23,     36,
     66,    161,    177,     24,     51,     82,    114,    209,    240,     37,
     98,    193,     67,    225,     25,    241,     38,     52,     68,     83,
    115,    130,     11,     12,     13,     14,     26,     27,     28,     29,
     30,     39,     40,     41,     42,     43,     44,     45,     46,     53,
     54,     55,     56,     57,     58,     59,     60,     61,     62,     69,
     70,     71,     72,     73,     74,     75,     76,     77,     78,     84,
     85,     86,     87,     88,     89,     90,     91,     92,     93,     94,
     99,    100,    101,    102,    103,    104,    105,    106,    107,    108,
    109,    110,    116,    117,    118,    119,    120,    121,    122,    123,
    124,    125,    126,    131,    132,    133,    134,    135,    136,    137,
    138,    139,    140,    141,    142,    146,    147,    148,    149,    150,
    151,    152,    153,    154,    155,    156,    157,    158,    162,    163,
    164,    165,    166,    167,    168,    169,    170,    171,    172,    173,
    174,    178,    179,    180,    181,    182,    183,    184,    185,    186,
    187,    188,    189,    190,    194,    195,    196,    197,    198,    199,
    200,    201,    202,    203,    204,    205,    206,    210,    211,    212,
    213,    214,    215,    216,    217,    218,    219,    220,    221,    222,
    226,    227,    228,    229,    230,    231,    232,    233,    234,    235,
    236,    237,    238,    242,    243,    244,    245,    246,    247,    248,
    249,    250,    251,    252,    253,  254,};
```

E.2 IR 12-bit tables for the q2 level

```
     16,     16,     54,    172,    554,    758,   1126,   4096
     16,     56,    160,    408,    616,    878,   3150,   4096
     54,    162,    448,    678,    914,   2032,   4096,   4096
    176,    414,    688,    954,   1612,   4096,   4096,   4096
    572,    638,    956,   1686,   4096,   4096,   4096,   4096
    722,    858,   2184,   4096,   4096,   4096,   4096,   4096
   1126,   3144,   4096,   4096,   4096,   4096,   4096,   4096
   4096,   4096,   4096,   4096,   4096,   4096,   4096,   4096
```

```
static const UINT8   dc_luminance_bits[17] =
{  0,  0,  2,  2,  3,  1,  1,  1,  1,
  1,  0,  3,  1,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{   6,   7,   5,   8,   3,   4,   9,   2,  10,   1,
    0,  11,  12,  13,  14,  15,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  1,  3,  3,  3,  3,  3,  3,
   3,  2,  1,  1,  1,  0,  0, 199,};


static const UINT8   ac_luminance_val[] =
{     1,     0,     2,     3,     4,     5,    17,     6,     7,    33,
      8,    18,    49,    19,    34,    65,    20,    81,    97,     9,
     50,   113,    21,    35,   129,    22,   145,   161,    66,   177,
     10,    23,    36,    51,   193,    82,   209,    24,    98,   114,
    225,   240,    37,    67,   130,   241,    38,    52,    25,   146,
     68,    83,    99,   162,   178,    11,    12,    13,    14,    26,
     27,    28,    29,    30,    39,    40,    41,    42,    43,    44,
     45,    46,    53,    54,    55,    56,    57,    58,    59,    60,
     61,    62,    69,    70,    71,    72,    73,    74,    75,    76,
     77,    78,    84,    85,    86,    87,    88,    89,    90,    91,
     92,    93,    94,   100,   101,   102,   103,   104,   105,   106,
    107,   108,   109,   110,   115,   116,   117,   118,   119,   120,
    121,   122,   123,   124,   125,   126,   131,   132,   133,   134,
    135,   136,   137,   138,   139,   140,   141,   142,   147,   148,
    149,   150,   151,   152,   153,   154,   155,   156,   157,   158,
    163,   164,   165,   166,   167,   168,   169,   170,   171,   172,
    173,   174,   179,   180,   181,   182,   183,   184,   185,   186,
    187,   188,   189,   190,   194,   195,   196,   197,   198,   199,
    200,   201,   202,   203,   204,   205,   206,   210,   211,   212,
    213,   214,   215,   216,   217,   218,   219,   220,   221,   222,
    226,   227,   228,   229,   230,   231,   232,   233,   234,   235,
    236,   237,   238,   242,   243,   244,   245,   246,   247,   248,
    249,   250,   251,   252,   253,  254,};
```

E.3 IR 12-bit tables for the q3 level

34,	34,	40,	64,	106,	176,	300,	532
34,	42,	58,	84,	130,	208,	346,	606
42,	58,	100,	144,	206,	310,	498,	844
68,	88,	148,	246,	376,	562,	878,	1442
118,	142,	222,	394,	684,	1110,	1774,	2886
208,	242,	354,	625,	1178,	2140,	3710,	4096
382,	436,	612,	1048,	2024,	3984,	4096,	4096
736,	830,	1134,	1876,	3590,	4096,	4096,	4096

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  2,  2,  3,  1,  1,  1,  1,
  0,  2,  3,  0,  0,  0,  0,  0,};


static const UINT8   dc_luminance_val[] =
{   5,   6,   4,   7,   2,   3,   8,   1,   0,   9,
  10,  11,  12,  13,  14,  15};


static const UINT8   ac_luminance_bits[17] =
{ 0,  0,  2,  1,  3,  3,  3,  3,  2,
  4,  4,  1,  1,  1,  0,  1, 197,};
```

```
static const UINT8   ac_luminance_val[] =
{      1,      2,      3,      0,      4,     17,      5,     18,     33,      6,
      49,     65,      7,     19,     81,     34,     97,     20,     50,    113,
     129,      8,     21,     35,     66,    145,     82,    161,    177,     22,
      51,    193,     36,     98,    209,    114,    225,      9,     23,     67,
     240,    241,     37,     52,     83,    130,     24,     38,     99,    115,
     146,    162,    178,    179,    131,     53,     68,     69,    147,     84,
     101,    194,    100,    116,     10,     11,     12,     13,     14,     25,
      26,     27,     28,     29,     30,     39,     40,     41,     42,     43,
      44,     45,     46,     54,     55,     56,     57,     58,     59,     60,
      61,     62,     70,     71,     72,     73,     74,     75,     76,     77,
      78,     85,     86,     87,     88,     89,     90,     91,     92,     93,
      94,    102,    103,    104,    105,    106,    107,    108,    109,    110,
     117,    118,    119,    120,    121,    122,    123,    124,    125,    126,
     132,    133,    134,    135,    136,    137,    138,    139,    140,    141,
     142,    148,    149,    150,    151,    152,    153,    154,    155,    156,
     157,    158,    163,    164,    165,    166,    167,    168,    169,    170,
     171,    172,    173,    174,    180,    181,    182,    183,    184,    185,
     186,    187,    188,    189,    190,    195,    196,    197,    198,    199,
     200,    201,    202,    203,    204,    205,    206,    210,    211,    212,
     213,    214,    215,    216,    217,    218,    219,    220,    221,    222,
     226,    227,    228,    229,    230,    231,    232,    233,    234,    235,
     236,    237,    238,    242,    243,    244,    245,    246,    247,    248,
     249,    250,    251,    252,    253,  254,};
```

E.4 IR 12-bit tables for the q4 level

```
      26,     26,     30,     50,     80,    136,    228,    402
      26,     32,     44,     64,     98,    158,    262,    460
      32,     44,     76,    108,    156,    234,    376,    640
      52,     66,    112,    186,    284,    426,    666,   1094
      90,    108,    168,    298,    518,    842,   1346,   2188
     158,    184,    268,    474,    894,   1622,   2814,   4096
     288,    330,    464,    794,   1534,   3022,   4096,   4096
     558,    630,    860,   1422,   2722,   4096,   4096,   4096
```

```
static const UINT8   dc_luminance_bits[17] =
{  0,  0,  1,  5,  1,  1,  1,  1,  1,
   0,  2,  3,  0,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{   6,   3,   4,   5,   7,   8,   2,   9,   1,   0,
   10,  11,  12,  13,  14,  15,};


static const UINT8   ac_luminance_bits[17] =
{  0,  0,  1,  3,  3,  3,  3,  3,  3,
   2,  4,  1,  1,  1,  0,  1, 197,};


static const UINT8   ac_luminance_val[] =
{     1,     2,     3,     4,     0,     5,    17,     6,    18,    33,
      7,    49,    65,    19,    34,    81,    20,    97,   113,     8,
     50,    21,    35,    66,   129,   145,   161,   177,    22,    51,
     82,   193,   209,     9,    36,    98,    67,   225,    23,    37,
    114,   240,   241,    52,    83,   130,    24,   146,    38,    99,
    115,   131,   179,   147,   162,   163,    25,    39,    68,   178,
     53,    69,    84,   116,   194,    10,    11,    12,    13,    14,
     26,    27,    28,    29,    30,    40,    41,    42,    43,    44,
     45,    46,    54,    55,    56,    57,    58,    59,    60,    61,
     62,    70,    71,    72,    73,    74,    75,    76,    77,    78,
     85,    86,    87,    88,    89,    90,    91,    92,    93,    94,
    100,   101,   102,   103,   104,   105,   106,   107,   108,   109,
    110,   117,   118,   119,   120,   121,   122,   123,   124,   125,
    126,   132,   133,   134,   135,   136,   137,   138,   139,   140,
    141,   142,   148,   149,   150,   151,   152,   153,   154,   155,
    156,   157,   158,   164,   165,   166,   167,   168,   169,   170,
    171,   172,   173,   174,   180,   181,   182,   183,   184,   185,
    186,   187,   188,   189,   190,   195,   196,   197,   198,   199,
    200,   201,   202,   203,   204,   205,   206,   210,   211,   212,
    213,   214,   215,   216,   217,   218,   219,   220,   221,   222,
    226,   227,   228,   229,   230,   231,   232,   233,   234,   235,
    236,   237,   238,   242,   243,   244,   245,   246,   247,   248,
    249,   250,   251,   252,   253,  254,};
```

E.5 IR 12-bit tables for the q5 level

16,	16,	16,	24,	38,	62,	100,	166
16,	16,	22,	30,	46,	70,	114,	186
18,	22,	36,	50,	70,	100,	154,	248
26,	32,	52,	82,	118,	168,	252,	392
42,	50,	76,	124,	198,	302,	460,	712
72,	82,	116,	188,	322,	538,	874,	1398
126,	142,	190,	300,	524,	940,	1652,	2814
230,	254,	332,	510,	886,	1638,	3068,	4096

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  2,  2,  3,  1,  1,  1,  1,
  1,  0,  3,  1,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{   6,   7,   5,   8,   3,   4,   9,   2,  10,   1,
   0,  11,  12,  13,  14,  15,};
```

```
static const UINT8   ac_luminance_bits[17] =
{   0,  0,  1,  3,  3,  3,  2,  5,  3,
  2,  4,  1,  1,  1,  0,  2, 195,};
```

```
static const UINT8   ac_luminance_val[] =
{     1,      2,      3,      4,      0,      5,     17,      6,     18,     33,
      7,     49,      8,     19,     34,     65,     81,     20,     97,    113,
     50,    129,      9,     21,     35,     66,    145,    161,     22,     36,
     51,     82,    177,    193,    209,     98,    225,     23,     67,    114,
    240,    241,     10,     37,     52,    130,     24,     83,    146,     38,
     68,    162,     99,     25,     53,     84,    115,    116,    131,     39,
     40,     69,    132,    178,     54,     85,    100,    147,    194,    210,
     11,     12,     13,     14,     26,     27,     28,     29,     30,     41,
     42,     43,     44,     45,     46,     55,     56,     57,     58,     59,
     60,     61,     62,     70,     71,     72,     73,     74,     75,     76,
     77,     78,     86,     87,     88,     89,     90,     91,     92,     93,
     94,    101,    102,    103,    104,    105,    106,    107,    108,    109,
    110,    117,    118,    119,    120,    121,    122,    123,    124,    125,
    126,    133,    134,    135,    136,    137,    138,    139,    140,    141,
    142,    148,    149,    150,    151,    152,    153,    154,    155,    156,
    157,    158,    163,    164,    165,    166,    167,    168,    169,    170,
    171,    172,    173,    174,    179,    180,    181,    182,    183,    184,
    185,    186,    187,    188,    189,    190,    195,    196,    197,    198,
    199,    200,    201,    202,    203,    204,    205,    206,    211,    212,
    213,    214,    215,    216,    217,    218,    219,    220,    221,    222,
    226,    227,    228,    229,    230,    231,    232,    233,    234,    235,
    236,    237,    238,    242,    243,    244,    245,    246,    247,    248,
    249,    250,    251,    252,    253,  254,};
```

E.6 IR 8-bit tables for the q1 level

```
     10,     10,     12,     19,     32,     52,     89,    158
     10,     12,     17,     25,     39,     62,    103,    181
     13,     17,     30,     43,     61,     92,    148,    251
     21,     26,     44,     73,    112,    167,    255,    255
     35,     43,     66,    117,    203,    255,    255,    255
     62,     72,    106,    186,    255,    255,    255,    255
    113,    130,    182,    255,    255,    255,    255,    255
    219,    247,    255,    255,    255,    255,    255,    255
```

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  2,  3,  1,  1,  1,  1,  0,
  3,  0,  0,  0,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{   3,   4,   1,   2,   5,   0,   6,   7,   8,   9,
   10,  11,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  2,  1,  4,  1,  4,  1,
    3,  3,  1,  1,  0,  1,  1, 137,};


static const UINT8   ac_luminance_val[] =
{      1,     2,     0,     3,    17,     4,    18,    33,    49,    65,
       5,    19,    34,    81,    97,    20,    50,   113,     6,    35,
     129,    66,   145,   161,    21,    82,   177,    36,    51,   193,
      98,   209,    22,    67,   114,   225,   240,     7,    52,   115,
     241,    83,   130,    37,    99,   146,    84,    68,   162,     8,
       9,    10,    23,    24,    25,    26,    38,    39,    40,    41,
      42,    53,    54,    55,    56,    57,    58,    69,    70,    71,
      72,    73,    74,    85,    86,    87,    88,    89,    90,   100,
     101,   102,   103,   104,   105,   106,   116,   117,   118,   119,
     120,   121,   122,   131,   132,   133,   134,   135,   136,   137,
     138,   147,   148,   149,   150,   151,   152,   153,   154,   163,
     164,   165,   166,   167,   168,   169,   170,   178,   179,   180,
     181,   182,   183,   184,   185,   186,   194,   195,   196,   197,
     198,   199,   200,   201,   202,   210,   211,   212,   213,   214,
     215,   216,   217,   218,   226,   227,   228,   229,   230,   231,
     232,   233,   234,   242,   243,   244,   245,   246,   247,   248,
     249,  250,};
```

E.7 IR 8-bit tables for the q2 level

7,	7,	8,	12,	20,	33,	55,	94
7,	8,	11,	16,	24,	38,	62,	107
8,	11,	19,	27,	38,	56,	88,	145
13,	17,	28,	45,	67,	98,	150,	241
22,	27,	41,	70,	117,	186,	255,	255
39,	45,	64,	109,	197,	255,	255,	255
69,	78,	108,	179,	255,	255,	255,	255
130,	146,	195,	255,	255,	255,	255,	255

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  2,  3,  1,  1,  1,  1,  0,
  3,  0,  0,  0,  0,  0,  0,  0,};


static const UINT8   dc_luminance_val[] =
{   3,   4,   1,   2,   5,   6,   0,   7,   8,   9,
  10,  11,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  2,  1,  3,  3,  3,  3,
  3,  3,  1,  1,  0,  2,  0, 135,};


static const UINT8   ac_luminance_val[] =
{      1,     2,     0,     3,    17,     4,    18,    33,     5,    49,
      65,    19,    34,    81,     6,    97,   113,    20,    50,   129,
      35,    66,   145,    21,   161,   177,     7,    82,   193,    36,
      51,    22,    98,   114,   209,   225,   240,   241,    52,    67,
     115,   130,    37,    83,    99,   146,   162,    23,    53,    68,
     178,     8,     9,    10,    24,    25,    26,    38,    39,    40,
      41,    42,    54,    55,    56,    57,    58,    69,    70,    71,
      72,    73,    74,    84,    85,    86,    87,    88,    89,    90,
     100,   101,   102,   103,   104,   105,   106,   116,   117,   118,
     119,   120,   121,   122,   131,   132,   133,   134,   135,   136,
     137,   138,   147,   148,   149,   150,   151,   152,   153,   154,
     163,   164,   165,   166,   167,   168,   169,   170,   179,   180,
     181,   182,   183,   184,   185,   186,   194,   195,   196,   197,
     198,   199,   200,   201,   202,   210,   211,   212,   213,   214,
     215,   216,   217,   218,   226,   227,   228,   229,   230,   231,
     232,   233,   234,   242,   243,   244,   245,   246,   247,   248,
     249,  250,};
```

E.8 IR 8-bit tables for the q3 level

10,	10,	10,	10,	10,	11,	15,	20
10,	10,	10,	10,	10,	12,	16,	21
10,	10,	10,	10,	11,	14,	18,	25
10,	10,	10,	12,	14,	18,	23,	32
11,	11,	12,	15,	19,	25,	32,	44
13,	14,	16,	20,	26,	35,	47,	63
18,	20,	22,	28,	37,	50,	68,	94
28,	29,	33,	41,	54,	74,	102,	143

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  2,  3,  1,  1,  1,  1,  0,
  3,  0,  0,  0,  0,  0,  0,  0,};
static const UINT8   dc_luminance_val[] =
{   3,   4,   1,   2,   5,   0,   6,   7,   8,   9,
  10,  11,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  1,  3,  3,  2,  5,  2,
  4,  5,  1,  1,  1,  1,  0, 131,};
```

```
static const UINT8   ac_luminance_val[] =
{     1,      2,     17,      0,      3,     33,      4,     18,     49,     65,
     81,      5,     19,     34,     97,    113,     50,    129,     66,    145,
    161,    177,     20,     35,    193,    209,    240,      6,     82,    225,
    241,     51,     98,     21,    114,     36,     67,    130,     52,     83,
    146,      7,     22,     99,    115,    162,    179,     37,     68,     84,
    116,    131,    178,    194,    147,    163,    100,      8,      9,     10,
     23,     24,     25,     26,     38,     39,     40,     41,     42,     53,
     54,     55,     56,     57,     58,     69,     70,     71,     72,     73,
     74,     85,     86,     87,     88,     89,     90,    101,    102,    103,
    104,    105,    106,    117,    118,    119,    120,    121,    122,    132,
    133,    134,    135,    136,    137,    138,    148,    149,    150,    151,
    152,    153,    154,    164,    165,    166,    167,    168,    169,    170,
    180,    181,    182,    183,    184,    185,    186,    195,    196,    197,
    198,    199,    200,    201,    202,    210,    211,    212,    213,    214,
    215,    216,    217,    218,    226,    227,    228,    229,    230,    231,
    232,    233,    234,    242,    243,    244,    245,    246,    247,    248,
    249,  250,};
```

E.9 IR 8-bit tables for the q4 level

```
     5,      5,      5,      5,      5,      6,      8,     10
     5,      5,      5,      5,      5,      6,      8,     11
     5,      5,      5,      5,      6,      7,      9,     13
     5,      5,      5,      6,      7,      9,     12,     17
     6,      6,      6,      8,     10,     13,     16,     22
     7,      7,      8,     10,     13,     18,     24,     32
     9,     10,     11,     14,     19,     25,     34,     47
    14,     15,     17,     22,     27,     37,     51,     72
```

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  2,  3,  1,  1,  1,  1,  1,
  1,  1,  0,  0,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{   4,   5,   2,   3,   6,   1,   7,   0,   8,   9,
  10,  11,};
```

```
static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  2,  0,  5,  3,  3,  2,
  3,  6,  3,  1,  1,  1,  1, 129,};


static const UINT8   ac_luminance_val[] =
{      1,      2,      3,     17,      0,      4,     18,     33,     49,      5,
      65,     81,     19,     34,     97,      6,    113,     20,     50,    129,
      35,     66,    145,    161,    177,    193,      7,     82,    209,    225,
     240,     21,     51,     98,    241,     36,    114,     67,     22,     52,
     130,     37,     83,    146,      8,    162,     23,     68,     99,    178,
      53,    115,    116,     38,     69,     84,    100,    131,    132,    148,
     194,    210,    147,    163,      9,     10,     24,     25,     26,     39,
      40,     41,     42,     54,     55,     56,     57,     58,     70,     71,
      72,     73,     74,     85,     86,     87,     88,     89,     90,    101,
     102,    103,    104,    105,    106,    117,    118,    119,    120,    121,
     122,    133,    134,    135,    136,    137,    138,    149,    150,    151,
     152,    153,    154,    164,    165,    166,    167,    168,    169,    170,
     179,    180,    181,    182,    183,    184,    185,    186,    195,    196,
     197,    198,    199,    200,    201,    202,    211,    212,    213,    214,
     215,    216,    217,    218,    226,    227,    228,    229,    230,    231,
     232,    233,    234,    242,    243,    244,    245,    246,    247,    248,
     249,  250,};
```

E.10 IR 8-bit tables for the q5 level

```
       1,      1,      1,      1,      1,      1,      1,      1
       1,      1,      1,      1,      1,      1,      1,      1
       1,      1,      1,      1,      1,      1,      1,      1
       1,      1,      1,      1,      1,      1,      1,      1
       1,      1,      1,      1,      1,      1,      1,      1
       1,      1,      1,      1,      1,      1,      1,      1
       1,      1,      1,      1,      1,      1,      1,      1
       1,      1,      1,      1,      1,      1,      1,      1


static const UINT8   dc_luminance_bits[17] =
{   0,  0,  2,  2,  3,  1,  1,  1,  1,
  1,  0,  0,  0,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{   6,   7,   5,   8,   3,   4,   9,   2,  10,   1,
   0,  11,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  2,  2,  2,  1,  3,  3,
  3,  3,  1,  1,  0,  1,  1, 137,};


static const UINT8   ac_luminance_val[] =
{      1,      2,      3,      4,      5,     17,      6,     18,     19,      7,
      33,     34,      0,     20,     49,      8,     35,     65,     21,     50,
      81,     97,      9,     22,     36,     51,     66,    113,    129,     82,
     145,    161,    177,    240,     23,     37,     52,     67,     98,    193,
     209,    225,    241,     10,     83,    114,     24,     38,    130,     53,
      68,     99,    146,     25,     39,    115,    162,    178,    194,     54,
      84,    100,    132,    210,     40,     85,    131,     55,     69,     86,
     133,    226,    242,     26,     41,     42,     56,     57,     58,     70,
      71,     72,     73,     74,     87,     88,     89,     90,    101,    102,
     103,    104,    105,    106,    116,    117,    118,    119,    120,    121,
     122,    134,    135,    136,    137,    138,    147,    148,    149,    150,
     151,    152,    153,    154,    163,    164,    165,    166,    167,    168,
     169,    170,    179,    180,    181,    182,    183,    184,    185,    186,
     195,    196,    197,    198,    199,    200,    201,    202,    211,    212,
     213,    214,    215,    216,    217,    218,    227,    228,    229,    230,
     231,    232,    233,    234,    243,    244,    245,    246,    247,    248,
     249,  250,};
```

E.11 SAR 12-bit tables for the q1 level

```
  200,   200,   670,  1285,  1940,  2805,  4096,   4096
  200,   720,  1211,  1723,  1989,  3476,  4096,   4096
  670,  1211,  1759,  1960,  3470,  4096,  4096,   4096
 1285,  1723,  1964,  3498,  4096,  4096,  4096,   4096
 1940,  1989,  3470,  4096,  4096,  4096,  4096,   4096
 2805,  3476,  4096,  4096,  4096,  4096,  4096,   4096
 4096,  4096,  4096,  4096,  4096,  4096,  4096,   4096
 4096,  4096,  4096,  4096,  4096,  4096,  4096,   4096
```

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  2,  3,  1,  1,  1,  1,  0,
  0,  7,  0,  0,  0,  0,  0,  0};


static const UINT8   dc_luminance_val[] =
{   3,   4,   1,   2,   5,   0,   6,   7,   8,   9,
  10,  11,  12,  13,  14,  15,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  1,  4,  1,  3,  3,  3,  3,
  3,  2,  1,  1,  0,  2,  0, 199,};


static const UINT8   ac_luminance_val[] =
{     1,     0,     2,     3,    17,    33,     4,    18,    49,     5,
     65,    81,    34,    97,   113,    19,   129,   145,    20,   161,
    177,     6,    50,   193,    21,    66,   209,   240,    35,   225,
    241,    82,    22,    51,    98,   114,   130,    36,    67,    52,
     83,   146,   178,     7,     8,     9,    10,    11,    12,    13,
     14,    23,    24,    25,    26,    27,    28,    29,    30,    37,
     38,    39,    40,    41,    42,    43,    44,    45,    46,    53,
     54,    55,    56,    57,    58,    59,    60,    61,    62,    68,
     69,    70,    71,    72,    73,    74,    75,    76,    77,    78,
     84,    85,    86,    87,    88,    89,    90,    91,    92,    93,
     94,    99,   100,   101,   102,   103,   104,   105,   106,   107,
    108,   109,   110,   115,   116,   117,   118,   119,   120,   121,
    122,   123,   124,   125,   126,   131,   132,   133,   134,   135,
    136,   137,   138,   139,   140,   141,   142,   147,   148,   149,
    150,   151,   152,   153,   154,   155,   156,   157,   158,   162,
    163,   164,   165,   166,   167,   168,   169,   170,   171,   172,
    173,   174,   179,   180,   181,   182,   183,   184,   185,   186,
    187,   188,   189,   190,   194,   195,   196,   197,   198,   199,
    200,   201,   202,   203,   204,   205,   206,   210,   211,   212,
    213,   214,   215,   216,   217,   218,   219,   220,   221,   222,
    226,   227,   228,   229,   230,   231,   232,   233,   234,   235,
    236,   237,   238,   242,   243,   244,   245,   246,   247,   248,
    249,   250,   251,   252,   253,  254,};
```

E.12 SAR 12-bit tables for the q2 level

```
  220,  220,  352,  625,  940, 1805, 3221, 4096
  220,  352,  500,  823, 1589, 2720, 4096, 4096
  352,  500,  830, 1680, 2805, 4096, 4096, 4096
  625,  823, 1680, 2798, 4096, 4096, 4096, 4096
  940, 1589, 2805, 4096, 4096, 4096, 4096, 4096
 1805, 2720, 4096, 4096, 4096, 4096, 4096, 4096
 3221, 4096, 4096, 4096, 4096, 4096, 4096, 4096
 4096, 4096, 4096, 4096, 4096, 4096, 4096, 4096
```

```
static const UINT8   dc_luminance_bits[17] =
{  0,  0,  2,  3,  1,  1,  1,  0,  0,
   7,  1,  0,  0,  0,  0,  0,  0,};

static const UINT8   dc_luminance_val[] =
{   3,   4,   1,   2,   5,   0,   6,   7,   8,   9,
   10,  11,  12,  13,  14,  15,};

static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  2,  1,  4,  1,  4,  1,
   3,  2,  1,  1,  0,  1,  0, 203,};
```

```
static const UINT8   ac_luminance_val[] =
{     1,      2,      3,     17,      0,      4,     18,     33,     49,     65,
      5,     19,     34,     81,     97,     50,    113,    129,     20,     35,
    145,      6,     66,    161,    177,    193,     21,     82,    209,    240,
     51,    225,     36,     98,    241,     22,     67,    114,     52,     83,
    130,    146,    162,      7,      8,      9,     10,     11,     12,     13,
     14,     23,     24,     25,     26,     27,     28,     29,     30,     37,
     38,     39,     40,     41,     42,     43,     44,     45,     46,     53,
     54,     55,     56,     57,     58,     59,     60,     61,     62,     68,
     69,     70,     71,     72,     73,     74,     75,     76,     77,     78,
     84,     85,     86,     87,     88,     89,     90,     91,     92,     93,
     94,     99,    100,    101,    102,    103,    104,    105,    106,    107,
    108,    109,    110,    115,    116,    117,    118,    119,    120,    121,
    122,    123,    124,    125,    126,    131,    132,    133,    134,    135,
    136,    137,    138,    139,    140,    141,    142,    147,    148,    149,
    150,    151,    152,    153,    154,    155,    156,    157,    158,    163,
    164,    165,    166,    167,    168,    169,    170,    171,    172,    173,
    174,    178,    179,    180,    181,    182,    183,    184,    185,    186,
    187,    188,    189,    190,    194,    195,    196,    197,    198,    199,
    200,    201,    202,    203,    204,    205,    206,    210,    211,    212,
    213,    214,    215,    216,    217,    218,    219,    220,    221,    222,
    226,    227,    228,    229,    230,    231,    232,    233,    234,    235,
    236,    237,    238,    242,    243,    244,    245,    246,    247,    248,
    249,    250,    251,    252,    253,  254,};
```

E.13 SAR 12-bit tables for the q3 level

```
    20,    20,    70,   185,   740,  1805,  4096,  4096
    20,    72,   171,   523,  1129,  2476,  4096,  4096
    70,   173,   559,  1060,  2670,  4048,  4096,  4096
   182,   523,  1084,  2798,  4096,  4096,  4096,  4096
   740,  1182,  2605,  4096,  4096,  4096,  4096,  4096
  1807,  2420,  4096,  4096,  4096,  4096,  4096,  4096
  4096,  4096,  4096,  4096,  4096,  4096,  4096,  4096
  4096,  4096,  4096,  4096,  4096,  4096,  4096,  4096
```

```
static const UINT8   dc_luminance_bits[17] =
{  0,  0,  2,  2,  3,  1,  1,  1,  1,
  1,  0,  3,  1,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{   6,   7,   5,   8,   3,   4,   9,   2,  10,   1,
   0,  11,  12,  13,  14,  15,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  1,  3,  3,  4,  1,  3,  3,
  3,  2,  1,  1,  0,  2,  0, 199,};


static const UINT8   ac_luminance_val[] =
{     1,       2,       3,       4,       0,       5,      17,       6,       7,      18,
     33,      49,       8,      19,      65,      20,      34,      81,       9,      50,
     97,      21,      35,     113,     129,      22,      66,     145,     161,     177,
     23,      36,     240,      51,      82,     193,     209,     225,      10,      24,
     37,     241,      67,      98,      38,      52,     114,      25,      53,      83,
     54,     130,     146,      11,      12,      13,      14,      26,      27,      28,
     29,      30,      39,      40,      41,      42,      43,      44,      45,      46,
     55,      56,      57,      58,      59,      60,      61,      62,      68,      69,
     70,      71,      72,      73,      74,      75,      76,      77,      78,      84,
     85,      86,      87,      88,      89,      90,      91,      92,      93,      94,
     99,     100,     101,     102,     103,     104,     105,     106,     107,     108,
    109,     110,     115,     116,     117,     118,     119,     120,     121,     122,
    123,     124,     125,     126,     131,     132,     133,     134,     135,     136,
    137,     138,     139,     140,     141,     142,     147,     148,     149,     150,
    151,     152,     153,     154,     155,     156,     157,     158,     162,     163,
    164,     165,     166,     167,     168,     169,     170,     171,     172,     173,
    174,     178,     179,     180,     181,     182,     183,     184,     185,     186,
    187,     188,     189,     190,     194,     195,     196,     197,     198,     199,
    200,     201,     202,     203,     204,     205,     206,     210,     211,     212,
    213,     214,     215,     216,     217,     218,     219,     220,     221,     222,
    226,     227,     228,     229,     230,     231,     232,     233,     234,     235,
    236,     237,     238,     242,     243,     244,     245,     246,     247,     248,
    249,     250,     251,     252,     253,  254,};
```

E.14 SAR 12-bit tables for the q4 level

15,	15,	28,	54,	105,	215,	469,	1081
15,	29,	47,	77,	140,	275,	585,	1321
29,	47,	100,	170,	288,	524,	1043,	2232
56,	81,	176,	394,	753,	1361,	2567,	4096
117,	154,	311,	787,	1878,	3884,	4096,	4096
254,	321,	601,	1511,	4096,	4096,	4096,	4096
595,	734,	1285,	3063,	4096,	4096,	4096,	4096
1496,	1806,	2999,	4096,	4096,	4096,	4096,	4096

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  2,  3,  1,  1,  1,  1,  1,
  1,  1,  0,  3,  1,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{   7,   8,   5,   6,   9,   4,   3,  10,   2,   1,
   0,  11,  12,  13,  14,  15,};
```

```
static const UINT8   ac_luminance_bits[17] =
{   0,  0,  1,  3,  3,  4,  0,  4,  4,
  5,  2,  1,  1,  1,  0,  2, 195,};
```

```
static const UINT8   ac_luminance_val[] =
{     1,      2,      3,      4,      5,      6,     17,      0,      7,     18,
     33,      8,     19,     34,     49,     20,     65,     81,     97,      9,
     21,     50,    113,    129,     35,     66,    145,     22,    161,     36,
     51,     82,    177,     10,     23,     98,    193,    209,    225,     24,
     37,     67,    114,    241,     52,     83,    130,    240,     38,    146,
     25,    162,     68,     99,     53,    178,     39,    115,     26,     84,
    194,    210,     11,     12,     13,     14,     27,     28,     29,     30,
     40,     41,     42,     43,     44,     45,     46,     54,     55,     56,
     57,     58,     59,     60,     61,     62,     69,     70,     71,     72,
     73,     74,     75,     76,     77,     78,     85,     86,     87,     88,
     89,     90,     91,     92,     93,     94,    100,    101,    102,    103,
    104,    105,    106,    107,    108,    109,    110,    116,    117,    118,
    119,    120,    121,    122,    123,    124,    125,    126,    131,    132,
    133,    134,    135,    136,    137,    138,    139,    140,    141,    142,
    147,    148,    149,    150,    151,    152,    153,    154,    155,    156,
    157,    158,    163,    164,    165,    166,    167,    168,    169,    170,
    171,    172,    173,    174,    179,    180,    181,    182,    183,    184,
    185,    186,    187,    188,    189,    190,    195,    196,    197,    198,
    199,    200,    201,    202,    203,    204,    205,    206,    211,    212,
    213,    214,    215,    216,    217,    218,    219,    220,    221,    222,
    226,    227,    228,    229,    230,    231,    232,    233,    234,    235,
    236,    237,    238,    242,    243,    244,    245,    246,    247,    248,
    249,    250,    251,    252,    253,  254,};
```

E.15 SAR 12-bit tables for the q5 level

```
     45,     46,     46,     62,     86,    110,    164,    204
     46,     44,     60,     88,     98,    138,    196,    254
     46,     60,     90,     94,    122,    194,    232,    302
     62,     88,     94,    126,    200,    248,    286,    392
     86,     98,    122,    200,    256,    280,    398,    661
    110,    138,    194,    248,    280,    404,    690,   1180
    164,    196,    232,    286,    398,    690,   1180,   2142
    204,    254,    302,    392,    661,   1180,   2142,   3802
```

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  2,  3,  1,  1,  1,  1,  1,
  1,  0,  2,  3,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{   5,   6,   3,   4,   7,   8,   2,   1,   0,   9,
  10,  11,  12,  13,  14,  15,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  2,  2,  1,  3,  3,  3,
  3,  2,  1,  1,  0,  1,  1, 201,};

static const UINT8   ac_luminance_val[] =
{     1,     2,     3,     4,     5,    17,    18,     0,     6,    33,
     19,    34,    49,     7,    20,    65,    35,    50,    81,    21,
     97,   113,    66,   129,     8,    22,    36,    51,    82,   145,
    161,    67,    98,   177,    37,   114,   193,    23,    52,   209,
     83,   130,   225,    38,   146,   240,   241,    99,    24,    53,
     68,   162,   178,    84,   115,     9,    39,    54,   131,    10,
     11,    12,    13,    14,    25,    26,    27,    28,    29,    30,
     40,    41,    42,    43,    44,    45,    46,    55,    56,    57,
     58,    59,    60,    61,    62,    69,    70,    71,    72,    73,
     74,    75,    76,    77,    78,    85,    86,    87,    88,    89,
     90,    91,    92,    93,    94,   100,   101,   102,   103,   104,
    105,   106,   107,   108,   109,   110,   116,   117,   118,   119,
    120,   121,   122,   123,   124,   125,   126,   132,   133,   134,
    135,   136,   137,   138,   139,   140,   141,   142,   147,   148,
    149,   150,   151,   152,   153,   154,   155,   156,   157,   158,
    163,   164,   165,   166,   167,   168,   169,   170,   171,   172,
    173,   174,   179,   180,   181,   182,   183,   184,   185,   186,
    187,   188,   189,   190,   194,   195,   196,   197,   198,   199,
    200,   201,   202,   203,   204,   205,   206,   210,   211,   212,
    213,   214,   215,   216,   217,   218,   219,   220,   221,   222,
    226,   227,   228,   229,   230,   231,   232,   233,   234,   235,
    236,   237,   238,   242,   243,   244,   245,   246,   247,   248,
    249,   250,   251,   252,   253,  254,};
```

E.16 Radar 8-bit tables for the q1 level

38,	38,	45,	54,	62,	74,	85,	150
38,	45,	52,	60,	69,	82,	130,	205
45,	52,	60,	68,	80,	130,	185,	220
54,	60,	68,	80,	130,	185,	200,	255
62,	69,	80,	130,	185,	200,	255,	255
74,	82,	130,	185,	200,	255,	255,	255
85,	130,	185,	200,	255,	255,	255,	255
150,	205,	220,	255,	255,	255,	255,	255

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  3,  1,  1,  1,  1,  0,  2,
  3,  0,  0,  0,  0,  0,  0,  0,};

static const UINT8   dc_luminance_val[] =
{   1,   2,   3,   0,   4,   5,   6,   7,   8,   9,
  10,  11,};

static const UINT8   ac_luminance_bits[17] =
{   0,  1,  0,  2,  2,  1,  3,  3,  2,
  5,  3,  1,  1,  0,  1,  2, 135,};
```

```
static const UINT8   ac_luminance_val[] =
{     1,      2,     17,      0,     33,     49,      3,     18,     65,     34,
     81,     97,    113,    129,      4,     19,     50,    145,    161,     66,
    177,    193,    209,    225,    240,     35,     82,    241,     20,     98,
      5,     51,    114,    130,     67,    146,     21,     36,    162,     83,
    178,    194,     52,      6,      7,      8,      9,     10,     22,     23,
     24,     25,     26,     37,     38,     39,     40,     41,     42,     53,
     54,     55,     56,     57,     58,     68,     69,     70,     71,     72,
     73,     74,     84,     85,     86,     87,     88,     89,     90,     99,
    100,    101,    102,    103,    104,    105,    106,    115,    116,    117,
    118,    119,    120,    121,    122,    131,    132,    133,    134,    135,
    136,    137,    138,    147,    148,    149,    150,    151,    152,    153,
    154,    163,    164,    165,    166,    167,    168,    169,    170,    179,
    180,    181,    182,    183,    184,    185,    186,    195,    196,    197,
    198,    199,    200,    201,    202,    210,    211,    212,    213,    214,
    215,    216,    217,    218,    226,    227,    228,    229,    230,    231,
    232,    233,    234,    242,    243,    244,    245,    246,    247,    248,
    249,  250,};
```

E.17 Radar 8-bit tables for the q2 level

```
     38,     38,     41,     44,     48,     50,     70,    140
     38,     43,     39,     46,     48,     50,    100,    190
     41,     39,     46,     47,     46,     70,    190,    220
     44,     46,     47,     46,     50,    150,    210,    240
     48,     48,     46,     50,    140,    200,    230,    255
     50,     50,     70,    150,    200,    230,    255,    255
     70,    100,    190,    210,    230,    255,    255,    255
    140,    190,    220,    240,    255,    255,    255,    255
```

```
static const UINT8   dc_luminance_bits[17] =
{  0,  0,  3,  1,  1,  1,  1,  0,  2,
   3,  0,  0,  0,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{  1,   2,   3,   0,   4,   5,   6,   7,   8,   9,
  10,  11,};
```

```
static const UINT8   ac_luminance_bits[17] =

{   0,  1,  0,  2,  1,  3,  4,  1,  3,
  2,  5,  1,  1,  0,  1,  2, 135,};


static const UINT8   ac_luminance_val[] =

{      1,       2,      17,      33,       0,      18,      49,       3,      34,      65,
      81,      97,      19,      50,     113,      66,     129,       4,      82,     145,
     161,     177,      35,     193,     209,     240,      98,     225,     114,     241,
      20,      51,      67,     130,       5,     146,      83,     162,      36,     178,
      21,     194,     210,      52,      99,     226,     115,     242,       6,       7,
       8,       9,      10,      22,      23,      24,      25,      26,      37,      38,
      39,      40,      41,      42,      53,      54,      55,      56,      57,      58,
      68,      69,      70,      71,      72,      73,      74,      84,      85,      86,
      87,      88,      89,      90,     100,     101,     102,     103,     104,     105,
     106,     116,     117,     118,     119,     120,     121,     122,     131,     132,
     133,     134,     135,     136,     137,     138,     147,     148,     149,     150,
     151,     152,     153,     154,     163,     164,     165,     166,     167,     168,
     169,     170,     179,     180,     181,     182,     183,     184,     185,     186,
     195,     196,     197,     198,     199,     200,     201,     202,     211,     212,
     213,     214,     215,     216,     217,     218,     227,     228,     229,     230,
     231,     232,     233,     234,     243,     244,     245,     246,     247,     248,
     249,  250,};
```

E.18 Radar 8-bit tables for the q3 level

```
      12,      12,      15,      20,      28,      35,      40,      60
      12,      15,      18,      22,      29,      37,      52,      86
      15,      18,      23,      29,      37,      48,      75,     106
      20,      22,      29,      36,      56,      68,      95,     130
      28,      29,      37,      56,      72,      92,     122,     188
      35,      37,      48,      68,      92,     128,     164,     255
      40,      52,      75,      95,     122,     164,     255,     255
      60,      86,     106,     130,     188,     255,     255,     255
```

```
static const UINT8   dc_luminance_bits[17] =

{   0,  0,  2,  3,  1,  1,  1,  1,  0,
  3,  0,  0,  0,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{  3,  4,  1,  2,  5,  0,  6,  7,  8,  9,
  10,  11,};

static const UINT8   ac_luminance_bits[17] =
{  0,  0,  2,  2,  1,  4,  1,  3,  3,
  2,  5,  1,  1,  0,  2,  0, 135,};


static const UINT8   ac_luminance_val[] =
{     1,     2,     3,    17,    33,     0,     4,    18,    49,    65,
     19,    34,    81,     5,    97,   113,    50,   129,    20,    35,
     66,   145,   161,   177,   193,     6,    82,   209,   225,   240,
     21,    36,    51,    98,   241,    67,   114,   130,    22,   146,
     52,    83,    37,   162,    68,   178,     7,    99,   194,   210,
    100,     8,     9,    10,    23,    24,    25,    26,    38,    39,
     40,    41,    42,    53,    54,    55,    56,    57,    58,    69,
     70,    71,    72,    73,    74,    84,    85,    86,    87,    88,
     89,    90,   101,   102,   103,   104,   105,   106,   115,   116,
    117,   118,   119,   120,   121,   122,   131,   132,   133,   134,
    135,   136,   137,   138,   147,   148,   149,   150,   151,   152,
    153,   154,   163,   164,   165,   166,   167,   168,   169,   170,
    179,   180,   181,   182,   183,   184,   185,   186,   195,   196,
    197,   198,   199,   200,   201,   202,   211,   212,   213,   214,
    215,   216,   217,   218,   226,   227,   228,   229,   230,   231,
    232,   233,   234,   242,   243,   244,   245,   246,   247,   248,
    249,  250,};
```

E.19 Radar 8-bit tables for the q4 level

```
      5,     5,     8,    10,    20,    29,    40,    60
      5,     9,    12,    17,    23,    32,    44,    75
      8,    12,    19,    25,    34,    50,    61,    95
     10,    17,    25,    38,    55,    68,    86,   120
     20,    23,    34,    55,    70,    91,   110,   150
     29,    32,    50,    68,    91,   128,   165,   175
     40,    44,    61,    86,   110,   165,   190,   255
     60,    75,    95,   120,   150,   175,   255,   255
```

```
static const UINT8   dc_luminance_bits[17] =
{  0,  0,  2,  3,  1,  1,  1,  1,  1,
  1,  1,  0,  0,  0,  0,  0,  0,};


static const UINT8   dc_luminance_val[] =
{   4,   5,   2,   3,   6,   7,   1,   0,   8,   9,
  10,  11,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  2,  1,  3,  3,  3,  2,
  4,  4,  3,  1,  1,  0,  2, 131,};


static const UINT8   ac_luminance_val[] =
{     1,     2,     3,    17,     4,     0,    18,    33,     5,    49,
     65,    19,    34,    81,     6,    97,    20,    50,   113,   129,
     35,   145,   161,   177,     7,    21,    66,   193,   209,    82,
    225,   240,    36,    51,    98,   241,    22,   114,    67,   130,
     37,     8,    23,    52,    83,   146,   162,    68,    99,   178,
    194,    38,    53,    24,    84,   210,     9,    10,    25,    26,
     39,    40,    41,    42,    54,    55,    56,    57,    58,    69,
     70,    71,    72,    73,    74,    85,    86,    87,    88,    89,
     90,   100,   101,   102,   103,   104,   105,   106,   115,   116,
    117,   118,   119,   120,   121,   122,   131,   132,   133,   134,
    135,   136,   137,   138,   147,   148,   149,   150,   151,   152,
    153,   154,   163,   164,   165,   166,   167,   168,   169,   170,
    179,   180,   181,   182,   183,   184,   185,   186,   195,   196,
    197,   198,   199,   200,   201,   202,   211,   212,   213,   214,
    215,   216,   217,   218,   226,   227,   228,   229,   230,   231,
    232,   233,   234,   242,   243,   244,   245,   246,   247,   248,
    249,  250,};
```

E.20 Radar 8-bit tables for the q5 level

8,	7,	7,	7,	8,	9,	11,	13
7,	7,	7,	7,	8,	9,	11,	14
7,	7,	8,	8,	9,	11,	13,	16
7,	7,	8,	10,	12,	14,	16,	20
8,	8,	9,	12,	15,	18,	22,	26
9,	9,	11,	14,	18,	23,	29,	36
11,	11,	13,	16,	22,	29,	38,	49
13,	14,	16,	20,	26,	36,	49,	65

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  1,  5,  1,  1,  1,  1,  1,
  1,  0,  0,  0,  0,  0,  0,  0,};
static const UINT8   dc_luminance_val[] =
{   0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
  10,  11,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  1,  3,  3,  2,  4,  3,
  5,  5,  4,  4,  0,  0,  1, 125, };
```

```
static const UINT8   ac_luminance_val[] =
{     1,      2,      3,      0,      4,     17,      5,     18,     33,     49,
     65,      6,     19,     81,     97,      7,     34,    113,     20,     50,
    129,    145,    161,      8,     35,     66,    177,    193,     21,     82,
    209,    240,     36,     51,     98,    114,    130,      9,     10,     22,
     23,     24,     25,     26,     37,     38,     39,     40,     41,     42,
     52,     53,     54,     55,     56,     57,     58,     67,     68,     69,
     70,     71,     72,     73,     74,     83,     84,     85,     86,     87,
     88,     89,     90,     99,    100,    101,    102,    103,    104,    105,
    106,    115,    116,    117,    118,    119,    120,    121,    122,    131,
    132,    133,    134,    135,    136,    137,    138,    146,    147,    148,
    149,    150,    151,    152,    153,    154,    162,    163,    164,    165,
    166,    167,    168,    169,    170,    178,    179,    180,    181,    182,
    183,    184,    185,    186,    194,    195,    196,    197,    198,    199,
    200,    201,    202,    210,    211,    212,    213,    214,    215,    216,
    217,    218,    225,    226,    227,    228,    229,    230,    231,    232,
    233,    234,    241,    242,    243,    244,    245,    246,    247,    248,
    249,  250,};
```

E.21 Visible 12-bit tables for the q1 level

```
   16,    16,    84,   372,   954,  2758,  4096,  4096
   16,    86,   260,   808,  2116,  4096,  4096,  4096
   84,   262,   848,  2178,  4096,  4096,  4096,  4096
  376,   814,  2188,  4096,  4096,  4096,  4096,  4096
  972,  1938,  4096,  4096,  4096,  4096,  4096,  4096
 2722,  4096,  4096,  4096,  4096,  4096,  4096,  4096
 4096,  4096,  4096,  4096,  4096,  4096,  4096,  4096
 4096,  4096,  4096,  4096,  4096,  4096,  4096,  4096
```

```
static const UINT8   dc_luminance_bits[17] =
{  0,  0,  1,  4,  3,  1,  1,  1,  1,
  0,  3,  1,  0,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{  7,   4,   5,   6,   8,   0,   3,   9,   2,   1,
  10,  11,  12,  13,  14,  15,};
```

```
static const UINT8   ac_luminance_bits[17] =
{   0,  0,  1,  3,  4,  1,  3,  2,  5,
  3,  2,  1,  1,  1,  0,  0, 199,};


static const UINT8   ac_luminance_val[] =
{     1,      0,      2,      3,      4,      5,      6,     17,      7,      8,
     18,     33,     19,     49,      9,     34,     65,    145,    240,     20,
     81,     98,     21,     50,     97,     22,     35,    113,     36,     66,
    129,     23,    193,     10,     51,     82,    161,    177,     24,     37,
    209,    225,    241,     25,     52,     67,     38,    114,     53,     83,
     11,     12,     13,     14,     26,     27,     28,     29,     30,     39,
     40,     41,     42,     43,     44,     45,     46,     54,     55,     56,
     57,     58,     59,     60,     61,     62,     68,     69,     70,     71,
     72,     73,     74,     75,     76,     77,     78,     84,     85,     86,
     87,     88,     89,     90,     91,     92,     93,     94,     99,    100,
    101,    102,    103,    104,    105,    106,    107,    108,    109,    110,
    115,    116,    117,    118,    119,    120,    121,    122,    123,    124,
    125,    126,    130,    131,    132,    133,    134,    135,    136,    137,
    138,    139,    140,    141,    142,    146,    147,    148,    149,    150,
    151,    152,    153,    154,    155,    156,    157,    158,    162,    163,
    164,    165,    166,    167,    168,    169,    170,    171,    172,    173,
    174,    178,    179,    180,    181,    182,    183,    184,    185,    186,
    187,    188,    189,    190,    194,    195,    196,    197,    198,    199,
    200,    201,    202,    203,    204,    205,    206,    210,    211,    212,
    213,    214,    215,    216,    217,    218,    219,    220,    221,    222,
    226,    227,    228,    229,    230,    231,    232,    233,    234,    235,
    236,    237,    238,    242,    243,    244,    245,    246,    247,    248,
    249,    250,    251,    252,    253,  254,};
```

E.22 Visible 12-bit tables for the q2 level

```
   20,    20,    70,   185,   540,  1805,  4096,  4096
   20,    72,   171,   423,   729,  1976,  4096,  4096
   70,   173,   429,   760,  2670,  4048,  4096,  4096
  182,   423,   784,  2798,  4096,  4096,  4096,  4096
  540,   582,  2605,  4096,  4096,  4096,  4096,  4096
 1807,  1920,  4096,  4096,  4096,  4096,  4096,  4096
 4096,  4096,  4096,  4096,  4096,  4096,  4096,  4096
 4096,  4096,  4096,  4096,  4096,  4096,  4096,  4096
```

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  1,  4,  3,  1,  1,  1,  1,
  0,  3,  1,  0,  0,  0,  0,  0,};


static const UINT8   dc_luminance_val[] =
{   7,   4,   5,   6,   8,   0,   3,   9,   2,   1,
  10,  11,  12,  13,  14,  15,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  1,  3,  3,  4,  1,  3,  2,
  5,  2,  1,  1,  1,  0,  0, 199,};


static const UINT8   ac_luminance_val[] =
{      1,      0,      2,      3,      4,      5,     17,      6,      7,     18,
      33,     49,      8,     19,     65,     34,     81,     20,     97,     98,
     145,    240,     21,     50,    113,      9,     35,    129,     66,     82,
     225,     22,     36,    161,     23,     51,    241,    114,    177,    193,
      24,     37,     67,    209,     52,     83,     10,     38,    130,     25,
      53,     68,     11,     12,     13,     14,     26,     27,     28,     29,
      30,     39,     40,     41,     42,     43,     44,     45,     46,     54,
      55,     56,     57,     58,     59,     60,     61,     62,     69,     70,
      71,     72,     73,     74,     75,     76,     77,     78,     84,     85,
      86,     87,     88,     89,     90,     91,     92,     93,     94,     99,
     100,    101,    102,    103,    104,    105,    106,    107,    108,    109,
     110,    115,    116,    117,    118,    119,    120,    121,    122,    123,
     124,    125,    126,    131,    132,    133,    134,    135,    136,    137,
     138,    139,    140,    141,    142,    146,    147,    148,    149,    150,
     151,    152,    153,    154,    155,    156,    157,    158,    162,    163,
     164,    165,    166,    167,    168,    169,    170,    171,    172,    173,
     174,    178,    179,    180,    181,    182,    183,    184,    185,    186,
     187,    188,    189,    190,    194,    195,    196,    197,    198,    199,
     200,    201,    202,    203,    204,    205,    206,    210,    211,    212,
     213,    214,    215,    216,    217,    218,    219,    220,    221,    222,
     226,    227,    228,    229,    230,    231,    232,    233,    234,    235,
     236,    237,    238,    242,    243,    244,    245,    246,    247,    248,
     249,    250,    251,    252,    253,  254,};
```

E.23 Visible 12-bit tables for the q3 level

34,	34,	40,	64,	106,	176,	300,	532
34,	42,	58,	84,	130,	208,	346,	606
42,	58,	100,	144,	206,	310,	498,	844
68,	88,	148,	246,	376,	562,	878,	1442
118,	142,	222,	394,	684,	1110,	1774,	2886
208,	242,	354,	625,	1178,	2140,	3710,	4096
382,	436,	612,	1048,	2024,	3984,	4096,	4096
736,	830,	1134,	1876,	3590,	4096,	4096,	4096

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  1,  4,  3,  1,  1,  1,  0,
  2,  3,  0,  0,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{   6,   3,   4,   5,   7,   0,   2,   8,   1,   9,
  10,  11,  12,  13,  14,  15,};
```

```
static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  1,  3,  3,  2,  5,  2,
  3,  6,  1,  1,  1,  1,  0, 195,};
```

```
static const UINT8   ac_luminance_val[] =
{     1,      2,      3,      0,      4,     17,      5,     18,     33,      6,
     49,      7,     19,     34,     65,     81,     20,     97,     50,    113,
    129,      8,     35,     66,    145,    163,    226,     21,    161,    177,
     36,     51,     52,     82,    100,    193,     22,     98,    209,     67,
     83,    114,    146,    210,    225,     68,     84,    115,    130,     23,
     37,    178,    240,    241,      9,     99,    162,     24,     38,     53,
    131,    194,     69,    147,     39,    195,     54,    116,    179,    211,
     10,     11,     12,     13,     14,     25,     26,     27,     28,     29,
     30,     40,     41,     42,     43,     44,     45,     46,     55,     56,
     57,     58,     59,     60,     61,     62,     70,     71,     72,     73,
     74,     75,     76,     77,     78,     85,     86,     87,     88,     89,
     90,     91,     92,     93,     94,    101,    102,    103,    104,    105,
    106,    107,    108,    109,    110,    117,    118,    119,    120,    121,
    122,    123,    124,    125,    126,    132,    133,    134,    135,    136,
    137,    138,    139,    140,    141,    142,    148,    149,    150,    151,
    152,    153,    154,    155,    156,    157,    158,    164,    165,    166,
    167,    168,    169,    170,    171,    172,    173,    174,    180,    181,
    182,    183,    184,    185,    186,    187,    188,    189,    190,    196,
    197,    198,    199,    200,    201,    202,    203,    204,    205,    206,
    212,    213,    214,    215,    216,    217,    218,    219,    220,    221,
    222,    227,    228,    229,    230,    231,    232,    233,    234,    235,
    236,    237,    238,    242,    243,    244,    245,    246,    247,    248,
    249,    250,    251,    252,    253,   254,};
```

E.24 Visible 12-bit tables for the q4 level

```
     18,     20,     28,     48,     82,    142,    258,    494
     20,     28,     42,     64,    102,    170,    306,    576
     28,     42,     76,    114,    172,    272,    466,    848
     50,     66,    118,    214,    348,    550,    910,   1588
     90,    112,    186,    364,    696,   1216,   2066,   3558
    168,    198,    312,    612,   1290,   2580,   4096,   4096
    328,    382,    574,   1086,   2356,   4096,   4096,   4096
    684,    786,   1140,   2066,   4096,   4096,   4096,   4096
```

```
static const UINT8   dc_luminance_bits[17] =
{  0,  0,  1,  4,  3,  1,  1,  1,  1,
  0,  3,  1,  0,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{   7,   4,   5,   6,   8,   0,   3,   9,   2,   1,
  10,  11,  12,  13,  14,  15,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  1,  3,  3,  3,  2,  4,  4,
  3,  5,  3,  1,  1,  2,  0, 191,};


static const UINT8   ac_luminance_val[] =
{      1,      2,      3,      4,      0,      5,     17,      6,     18,     33,
       7,     49,     19,     34,     65,     81,      8,     20,     97,    113,
      21,     50,    129,     35,     66,    145,    163,    227,      9,     36,
      82,    161,    177,     22,     51,     52,     98,    193,     67,    100,
     114,    209,     23,     37,     68,     83,     84,    115,    146,    210,
     225,    130,    131,    178,     99,    147,    241,     24,    162,    240,
      10,     38,    179,     25,     53,     69,    195,    226,     39,    194,
      54,    132,    211,     11,     12,     13,     14,     26,     27,     28,
      29,     30,     40,     41,     42,     43,     44,     45,     46,     55,
      56,     57,     58,     59,     60,     61,     62,     70,     71,     72,
      73,     74,     75,     76,     77,     78,     85,     86,     87,     88,
      89,     90,     91,     92,     93,     94,    101,    102,    103,    104,
     105,    106,    107,    108,    109,    110,    116,    117,    118,    119,
     120,    121,    122,    123,    124,    125,    126,    133,    134,    135,
     136,    137,    138,    139,    140,    141,    142,    148,    149,    150,
     151,    152,    153,    154,    155,    156,    157,    158,    164,    165,
     166,    167,    168,    169,    170,    171,    172,    173,    174,    180,
     181,    182,    183,    184,    185,    186,    187,    188,    189,    190,
     196,    197,    198,    199,    200,    201,    202,    203,    204,    205,
     206,    212,    213,    214,    215,    216,    217,    218,    219,    220,
     221,    222,    228,    229,    230,    231,    232,    233,    234,    235,
     236,    237,    238,    242,    243,    244,    245,    246,    247,    248,
     249,    250,    251,    252,    253,  254,};
```

E.25 Visible 12-bit tables for the q5 level

13,	13,	15,	25,	40,	68,	114,	161
13,	16,	22,	32,	49,	79,	131,	200
16,	22,	38,	54,	78,	117,	198,	245
26,	33,	56,	93,	122,	206,	235,	255
45,	54,	84,	139,	205,	230,	255,	255
65,	92,	134,	207,	235,	255,	255,	255
112,	165,	198,	245,	255,	255,	255,	255
190,	205,	250,	255,	255,	255,	255,	255

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  0,  6,  3,  1,  1,  1,  0,
  3,  1,  0,  0,  0,  0,  0,  0,};
static const UINT8   dc_luminance_val[] =
{   4,   5,   6,   7,   8,   9,   0,   2,   3,  10,
   1,  11,  12,  13,  14,  15,};


static const UINT8   ac_luminance_bits[17] =
{   0,  0,  1,  3,  2,  5,  2,  4,  3,
  4,  7,  3,  2,  0,  0,  1, 189,};
```

```
static const UINT8   ac_luminance_val[] =
{     1,      2,      3,      4,      5,     17,      0,      6,      7,     18,
     33,     49,     65,      8,     19,     34,     81,     20,     97,    113,
      9,     21,     50,    129,     35,     66,    145,    161,    177,    193,
    228,     22,     82,    209,     36,     37,     51,     84,     98,    164,
    225,    240,     23,     52,     53,     67,    100,    114,    241,     10,
     68,    116,    132,    148,    212,     24,     69,    130,    146,     38,
     83,     85,     99,    101,    178,    162,    115,    147,     25,     39,
    131,    180,    194,    196,     54,    179,    210,     26,    163,     11,
     12,     13,     14,     27,     28,     29,     30,     40,     41,     42,
     43,     44,     45,     46,     55,     56,     57,     58,     59,     60,
     61,     62,     70,     71,     72,     73,     74,     75,     76,     77,
     78,     86,     87,     88,     89,     90,     91,     92,     93,     94,
    102,    103,    104,    105,    106,    107,    108,    109,    110,    117,
    118,    119,    120,    121,    122,    123,    124,    125,    126,    133,
    134,    135,    136,    137,    138,    139,    140,    141,    142,    149,
    150,    151,    152,    153,    154,    155,    156,    157,    158,    165,
    166,    167,    168,    169,    170,    171,    172,    173,    174,    181,
    182,    183,    184,    185,    186,    187,    188,    189,    190,    195,
    197,    198,    199,    200,    201,    202,    203,    204,    205,    206,
    211,    213,    214,    215,    216,    217,    218,    219,    220,    221,
    222,    226,    227,    229,    230,    231,    232,    233,    234,    235,
    236,    237,    238,    242,    243,    244,    245,    246,    247,    248,
    249,    250,    251,    252,    253,  254,};
```

E.26 Visible-8 bit table for the q1 level

```
     13,     13,     15,     25,     40,     68,    114,    201
     13,     16,     22,     32,     49,     79,    131,    230
     16,     22,     38,     54,     78,    117,    188,    255
     26,     33,     56,     93,    142,    213,    255,    255
     45,     54,     84,    149,    255,    255,    255,    255
     79,     92,    134,    237,    255,    255,    255,    255
    144,    165,    232,    255,    255,    255,    255,    255
    255,    255,    255,    255,    255,    255,    255,    255
```

```
static const UINT8   ac_luminance_bits[17] =
{  0,  0,  2,  2,  1,  4,  1,  3,  3,
  3,  3,  1,  1,  0,  1,  2, 135,};
```

```
static const UINT8   ac_luminance_val[] =
{     1,      2,      0,     17,      3,      4,     18,     33,     49,     65,
      5,     19,     81,     34,     97,    113,     50,    161,    225,     20,
    129,    145,     35,     66,      6,     21,     51,     82,    177,     98,
    193,    209,    240,     36,     67,     83,    114,    241,     22,     52,
    178,    130,    162,    194,     68,    146,     37,     84,     99,      7,
      8,      9,     10,     23,     24,     25,     26,     38,     39,     40,
     41,     42,     53,     54,     55,     56,     57,     58,     69,     70,
     71,     72,     73,     74,     85,     86,     87,     88,     89,     90,
    100,    101,    102,    103,    104,    105,    106,    115,    116,    117,
    118,    119,    120,    121,    122,    131,    132,    133,    134,    135,
    136,    137,    138,    147,    148,    149,    150,    151,    152,    153,
    154,    163,    164,    165,    166,    167,    168,    169,    170,    179,
    180,    181,    182,    183,    184,    185,    186,    195,    196,    197,
    198,    199,    200,    201,    202,    210,    211,    212,    213,    214,
    215,    216,    217,    218,    226,    227,    228,    229,    230,    231,
    232,    233,    234,    242,    243,    244,    245,    246,    247,    248,
    249,  250,};


static const UINT8      dc_luminance_bits[17] =
{   0,  0,  2,  3,  1,  1,  1,  1,  0,
 3,  0,  0,  0,  0,  0,  0,  0,};


static const UINT8      dc_luminance_val[] =
{ 2,   3,   0,   1,   4,   5,   6,   7,   8,   9,
 10,  11,};
```

E.27 Visible 8-bit table for the q2 level

8,	8,	8,	12,	19,	31,	50,	83
8,	8,	11,	15,	23,	35,	57,	93
9,	11,	18,	25,	35,	50,	77,	124
13,	16,	26,	41,	59,	84,	126,	196
21,	25,	38,	62,	99,	151,	230,	255
36,	41,	58,	94,	161,	255,	255,	255
63,	71,	95,	150,	255,	255,	255,	255
115,	127,	166,	255,	255,	255,	255,	255

```
static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  2,  1,  3,  3,  3,  2,
  4,  5,  1,  1,  1,  0,  1, 133,};


static const UINT8   ac_luminance_val[] =
{      1,      2,      0,      3,     17,      4,     18,     33,      5,     49,
     65,     19,     34,     81,     97,    113,      6,     20,     50,    129,
     35,     98,    145,    161,    225,     66,    162,    177,     21,     82,
    193,    209,     51,     36,     67,    114,      7,     22,    240,     52,
     83,    130,    241,     37,     99,    146,    194,     23,     53,     68,
    115,    178,     84,    179,      8,      9,     10,     24,     25,     26,
     38,     39,     40,     41,     42,     54,     55,     56,     57,     58,
     69,     70,     71,     72,     73,     74,     85,     86,     87,     88,
     89,     90,    100,    101,    102,    103,    104,    105,    106,    116,
    117,    118,    119,    120,    121,    122,    131,    132,    133,    134,
    135,    136,    137,    138,    147,    148,    149,    150,    151,    152,
    153,    154,    163,    164,    165,    166,    167,    168,    169,    170,
    180,    181,    182,    183,    184,    185,    186,    195,    196,    197,
    198,    199,    200,    201,    202,    210,    211,    212,    213,    214,
    215,    216,    217,    218,    226,    227,    228,    229,    230,    231,
    232,    233,    234,    242,    243,    244,    245,    246,    247,    248,
    249,  250,};


static const UINT8      dc_luminance_bits[17] =
{   0,  0,  1,  5,  1,  1,  1,  0,  3,
  0,  0,  0,  0,  0,  0,  0,  0, };


static const UINT8      dc_luminance_val[] =
{  3,   0,   1,   2,   4,   5,   6,   7,   8,   9,
  10,  11, };
```

E.28 Visible 8-bit tables for the q3 level

10,	10,	10,	10,	10,	11,	15,	20
10,	10,	10,	10,	10,	12,	16,	21
10,	10,	10,	10,	11,	14,	18,	25
10,	10,	10,	12,	14,	18,	23,	32
11,	11,	12,	15,	19,	25,	32,	44
13,	14,	16,	20,	26,	35,	47,	63
18,	20,	22,	28,	37,	50,	68,	94
28,	29,	33,	41,	54,	74,	102,	143

```
static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  1,  3,  3,  2,  5,  2,
  2,  7,  5,  1,  2,  0,  0, 127,};
```

```
static const UINT8   ac_luminance_val[] =
{     1,     2,    17,     0,     3,    33,     4,    18,    49,    65,
     81,     5,    19,    34,    97,   113,    50,   129,   145,   161,
     20,    35,    66,    82,   177,   193,   209,     6,    98,    99,
    163,   225,   227,   240,    21,    51,   241,    36,    67,   114,
    130,   146,   211,    83,   115,    52,    68,   131,   162,   178,
    179,    22,     7,    37,   147,   194,   195,   210,    53,    84,
    100,    23,    69,     8,     9,    10,    24,    25,    26,    38,
     39,    40,    41,    42,    54,    55,    56,    57,    58,    70,
     71,    72,    73,    74,    85,    86,    87,    88,    89,    90,
    101,   102,   103,   104,   105,   106,   116,   117,   118,   119,
    120,   121,   122,   132,   133,   134,   135,   136,   137,   138,
    148,   149,   150,   151,   152,   153,   154,   164,   165,   166,
    167,   168,   169,   170,   180,   181,   182,   183,   184,   185,
    186,   196,   197,   198,   199,   200,   201,   202,   212,   213,
    214,   215,   216,   217,   218,   226,   228,   229,   230,   231,
    232,   233,   234,   242,   243,   244,   245,   246,   247,   248,
    249,  250,};
```

```
static const UINT8      dc_luminance_bits[17] =
{  0,  0,  1,  5,  1,  1,  1,  0,  3,
  0,  0,  0,  0,  0,  0,  0,  0,};
```

```
static const UINT8      dc_luminance_val[] =
{  3,   0,   1,   2,   4,   5,   6,   7,   8,   9,
  10,  11,};
```

E.29 Visible 8-bit table for the q4 level

4,	4,	4,	4,	4,	5,	6,	7
4,	4,	4,	4,	5,	5,	6,	8
4,	4,	4,	5,	5,	6,	7,	9
4,	4,	5,	6,	6,	8,	9,	11
4,	5,	5,	6,	8,	10,	12,	14
5,	5,	6,	8,	10,	13,	16,	20
6,	6,	7,	9,	12,	16,	21,	27
7,	8,	9,	11,	14,	20,	27,	36

```
static const UINT8   dc_luminance_bits[17] =
{   0,  0,  1,  5,  1,  1,  1,  1,  1,
  1,  0,  0,  0,  0,  0,  0,  0,};
```

```
static const UINT8   dc_luminance_val[] =
{  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
 10,  11,};
```

```
static const UINT8   ac_luminance_bits[17] =
{   0,  0,  2,  1,  3,  3,  2,  4,  3,
  5,  5,  4,  4,  0,  0,  1, 125, };
```

```
static const UINT8   ac_luminance_val[] =
{     1,      2,      3,      0,      4,     17,      5,     18,     33,     49,
     65,      6,     19,     81,     97,      7,     34,    113,     20,     50,
    129,    145,    161,      8,     35,     66,    177,    193,     21,     82,
    209,    240,     36,     51,     98,    114,    130,      9,     10,     22,
     23,     24,     25,     26,     37,     38,     39,     40,     41,     42,
     52,     53,     54,     55,     56,     57,     58,     67,     68,     69,
     70,     71,     72,     73,     74,     83,     84,     85,     86,     87,
     88,     89,     90,     99,    100,    101,    102,    103,    104,    105,
    106,    115,    116,    117,    118,    119,    120,    121,    122,    131,
    132,    133,    134,    135,    136,    137,    138,    146,    147,    148,
    149,    150,    151,    152,    153,    154,    162,    163,    164,    165,
    166,    167,    168,    169,    170,    178,    179,    180,    181,    182,
    183,    184,    185,    186,    194,    195,    196,    197,    198,    199,
    200,    201,    202,    210,    211,    212,    213,    214,    215,    216,
    217,    218,    225,    226,    227,    228,    229,    230,    231,    232,
    233,    234,    241,    242,    243,    244,    245,    246,    247,    248,
    249,  250,};
```

E.30 Visible 8-bit table for the q5 level

```
      1,      1,      1,      1,      1,      1,      1,      1
      1,      1,      1,      1,      1,      1,      1,      1
      1,      1,      1,      1,      1,      1,      1,      1
      1,      1,      1,      1,      1,      1,      1,      1
      1,      1,      1,      1,      1,      1,      1,      1
      1,      1,      1,      1,      1,      1,      1,      1
      1,      1,      1,      1,      1,      1,      1,      1
      1,      1,      1,      1,      1,      1,      1,      1
```

```
static const UINT8   ac_luminance_bits[17] =
{  0,  0,  2,  2,  2,  2,  1,  3,  3,
  2,  5,  1,  1,  0,  1,  2, 135,};
```

```
static const UINT8   ac_luminance_val[] =
{     1,      2,      3,      4,      5,     17,      6,     18,     33,      0,
      7,     19,     20,     34,     49,      8,     65,     21,     35,     50,
     81,     97,     66,    113,      9,     22,     36,    129,     51,     82,
    145,     23,    161,    177,    240,     24,     37,     38,     67,     98,
    193,    209,     52,     54,    114,    225,    241,     10,     40,     56,
     70,     83,    130,     72,     39,     53,     68,     86,     88,     99,
    102,    146,    104,    162,    166,    178,    194,     25,     55,     71,
     84,    115,    120,    134,    150,    232,     69,     85,     87,    118,
    136,    200,    226,    103,    152,    216,    131,    132,    168,    210,
    100,    116,    184,    242,     26,     41,     42,     57,     58,     73,
     74,     89,     90,    101,    105,    106,    117,    119,    121,    122,
    133,    135,    137,    138,    147,    148,    149,    151,    153,    154,
    163,    164,    165,    167,    169,    170,    179,    180,    181,    182,
    183,    185,    186,    195,    196,    197,    198,    199,    201,    202,
    211,    212,    213,    214,    215,    217,    218,    227,    228,    229,
    230,    231,    233,    234,    243,    244,    245,    246,    247,    248,
    249,  250,};


static const UINT8      dc_luminance_bits[17] =
{   0,  0,  1,  4,  3,  1,  1,  1,  1,
   0,  0,  0,  0,  0,  0,  0,  0,};

static const UINT8      dc_luminance_val[] =
{  7,   4,   5,   6,   8,   2,   3,   9,   0,   1,
   10,  11, };
```

APPENDIX F ENGINEERING DESIGN DETAILS FOR THE DOWNSAMPLE JPEG SYSTEM

F.1 INTRODUCTION

This appendix provides information concerning the engineering principles behind the Downsample JPEG algorithm. An understanding of the material presented in this appendix is not mandatory for implementation. Full implementation details are discussed in the normative section (Refer to Section 3.0, NITFS Lossy JPEG with Downsampling). The engineering details, however, do provide insight on the key components of the algorithm that directly influence the quality of the reconstructed image.

All document references made in this appendix apply to the NITFS Downsample JPEG Image Compression Standard. This standard is found in Section 5.0 of this document.

F.2 DOWNSAMPLE JPEG SYSTEM MODEL

The Downsample JPEG compression algorithm achieves very low bit rate compression using the scheme shown in Figure F-1. Decimation of the original image is used to achieve bit rates beyond what JPEG can accomplish alone (0.5-0.8 bits/pixel) due to the fixed 8x8 block size encoding structure. In this algorithm, the adverse effects of downsampling (e.g., aliasing and blurring) are traded-off with JPEG artifacts (e.g., blocking) by adjusting the relative compression contributions from each module. The quality of the reconstructed image after JPEG decompression and upsampling has been demonstrated to be competitive with several "state-of-the-art" low bit rate compression algorithms.

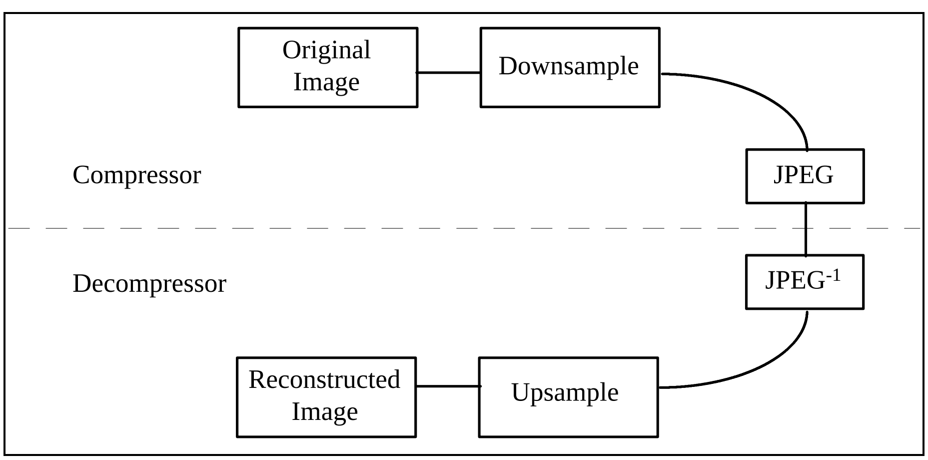

FIGURE F-1 System model for Downsample JPEG compression.

F.2.1 Downsampling model

Conceptual downsamping is performed using the model shown in Figure F-2. Note that all references to *t* refer to the continuous domain (e.g., time or space). $H_d(j\Omega)$ represents the ideal downsampling filter that is applied to the digital data, T_o is the old sampling period, T is the new sampling period, and R is the downsample ratio (Refer to the downsample ratio for either the row or column dimension as discussed in Section 5.1.1). Without loss of generality, T_o is assumed to be equal to one. The new sampling period, T, is then equal to the downsample ratio, R. R is constrained to be greater than one, so the new sample period is larger than the old (new sampling rate is smaller than the original).

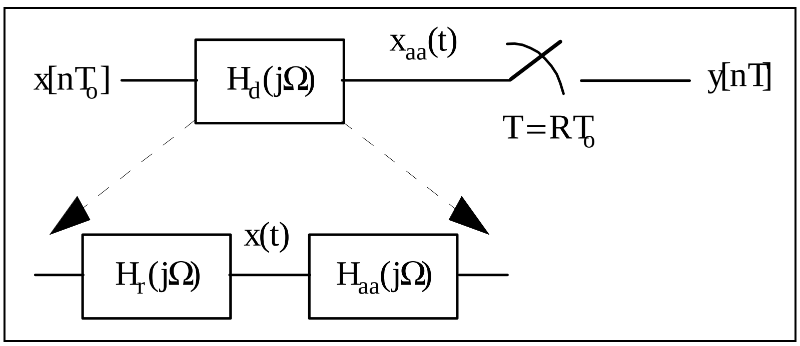

FIGURE F-2 Downsampling model.

$H_d(j\Omega)$ can conceptually be separated into two filters: an ideal reconstruction filter for discrete-to-continuous (D/C) conversion, and an ideal anti-aliasing filter. The reconstruction filter converts the digital signal to a continuous form, while the anti-aliasing filter prepares the continuous signal for resampling at a lower rate. The frequency responses of the two ideal filters are shown in Figure F-3.

$H_d(j\Omega)$ can conceptually be separated into two filters: an ideal reconstruction filter for discrete-to-continuous (D/C) conversion, and an ideal anti-aliasing filter. The reconstruction filter converts the digital signal to a continuous form, while the anti-aliasing filter prepares the continuous signal for resampling at a lower rate. The frequency responses of the two ideal filters are shown in Figure F-3.

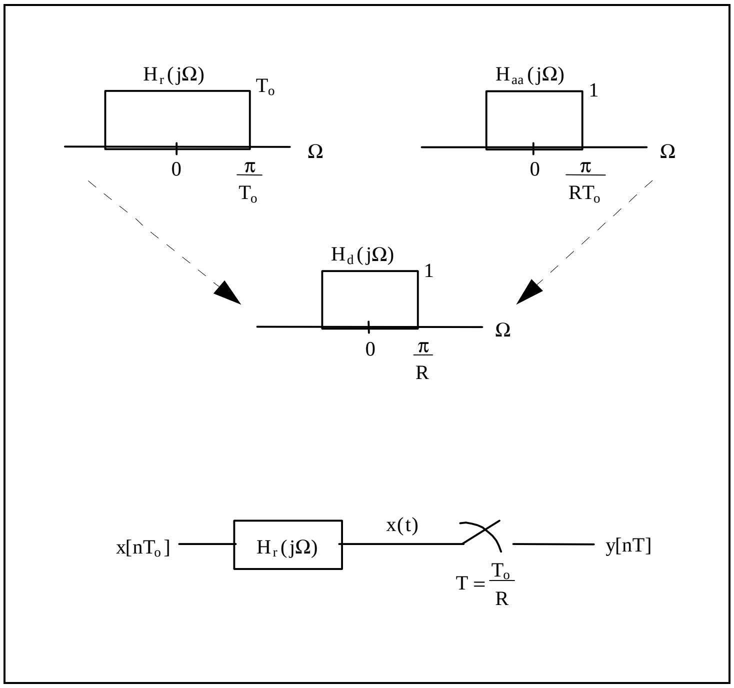

FIGURE F-3 Frequency responses of the ideal downsample filters.

The combined response, $H_d(j\Omega)$, is achieved by multiplying the reconstruction filter and anti-aliasing filter frequency responses with T_o equal to one. The resulting downsample filter performs both D/C conversion and anti-aliasing. The impulse response of the downsample filter is the sinc function defined by the following equation:

Ideal downsample filter impulse response:

$$h_d(t) = \frac{1}{R}\,\mathrm{sinc}\left(\frac{t}{R}\right) = \frac{\sin\left(\frac{\pi t}{R}\right)}{\pi t} \qquad -\infty < t < \infty$$

The sampler shown in Figure C2 converts the continuous signal back to the digital domain, but at the new sampling rate. The downsample ratio, R, in this system can be any floating point value greater than one.

F.2.2 Practical considerations for downsampling

The ideal filter can only be approximated due to its infinite length. The approximation is accomplished using a window of finite length to truncate the ideal impulse response. The window shape defines how abruptly the truncation is performed. This allows a trade-off between transition width and stopband attenuation of the windowed filter frequency response. The window to be used for downsampling is defined as follows:

Downsample window:

$$g_d(t) = \begin{cases} \sqrt{\cos\left(\dfrac{\pi t}{\alpha R}\right)} & -(\alpha R)/2 \le t \le (\alpha R)/2 \\ 0 & \text{else} \end{cases}$$

where, α is a parameter that specifies a *base* filter length. The actual filter length is the base length multiplied by the downsample ratio, R, as shown by the quantity in the denominator of the cosine argument. The filter length dependency on the downsample ratio gives the filter the desirable property of having a sharper cutoff in the frequency domain as the downsample ratio is increased. In effect, the transition width of the filter frequency response is kept proportional to the passband width. Sharper cutoff and lower stopband attenuation are issues that are more critical when the desired cutoff of the downsample filter decreases, since the signal energy tends to be concentrated in the low region of the frequency spectrum. The downsampled image is more susceptible to aliasing and severe distortion when the signal energy is high in the filter cutoff region.

The windowed downsample filter is found by multiplying the window with the ideal impulse response. The result is an equation that is similar to the un-normalized coefficients, c_{ij},. The ideal impulse response and the truncating window are illustrated in Figure F-4.

In the conceptual model, the filtering and sampling are two separate, sequential operations. Therefore, the filtering operation first reconstructs an infinite number of values in the process of changing the input from discrete to continuous. Then the sampler saves only a small subset of values and discards the rest. In practice, these operations can be combined by performing the filtering calculations at only the locations that will actually be sampled. This complexity reduction is reflected in the implementation details of the normative sections.

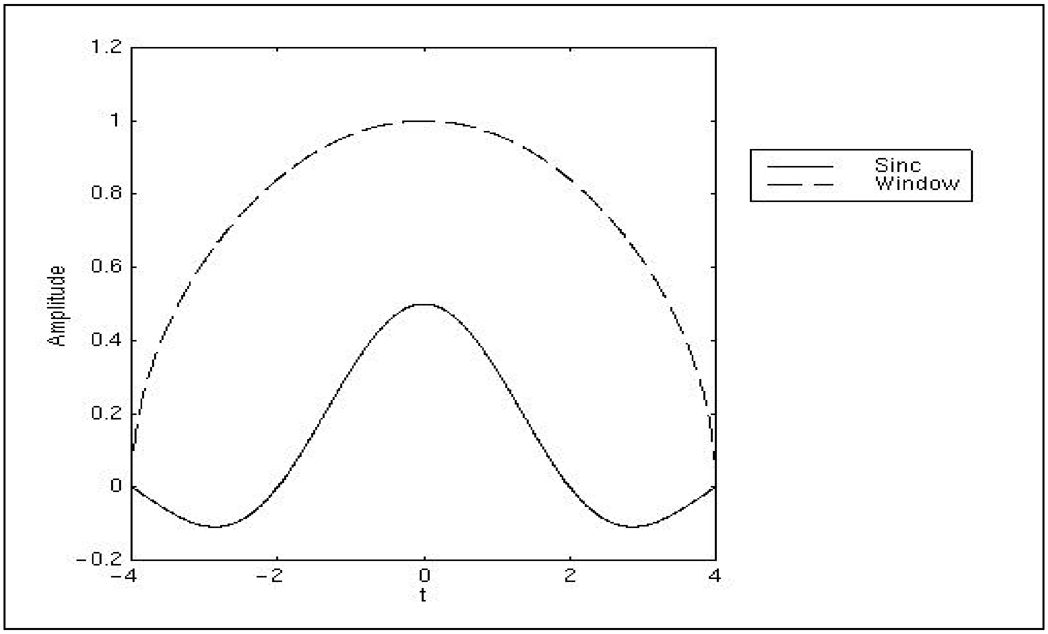

FIGURE F-4 Ideal downsample filter and truncating window.

F.2.3 Upsampling model

Figure F-5 shows the conceptual upsampling process. $H_r(j\Omega)$ represents the ideal reconstruction filter that is used to perform D/C conversion. Once the signal is converted to the continuous domain, it is resampled with a new sampling period, T. The upsampling ratio, R is defined to be greater than one, so that the new sample period is smaller than the old period, T_o (or new sampling rate is higher than the original). Similar to the downsample case, T_o is assumed to be equal to one without loss of generality. The ideal frequency response is shown in Figure F-6.

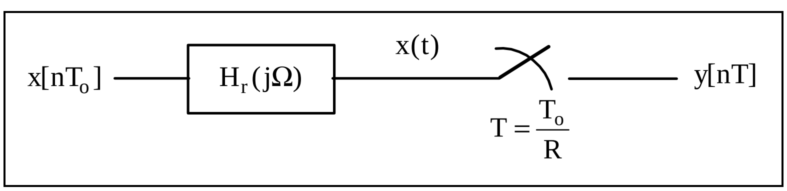

FIGURE F-5 Upsampling model.

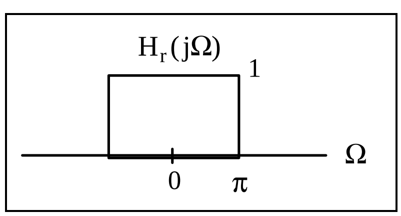

FIGURE F-6 Frequency response of the ideal upsampling filter.

The impulse response of the upsample filter is the sinc function defined by the following equation:

Ideal upsample filter impulse response:

$$h_r(t) = \mathrm{sinc}(t) = \frac{\sin(\pi t)}{\pi t} \qquad -\infty < t < \infty$$

The sampler shown in Figure F-5 converts the continuous signal back to the digital domain, but at the new sampling rate. The upsample ratio, R, in this system has the potential to be any floating point value greater than one.

F.2.4 Practical considerations for upsampling

Similar to the downsampling situation, the ideal upsample impulse response is truncated using a window in order to achieve a practical filter. The window that is used to form the practical upsample filter is defined as follows:

Upsample window:

$$g_u(t) = \begin{cases} \left(\cos\left(\dfrac{\pi t}{\alpha}\right)\right)^2 & -\alpha/2 \le t \le \alpha/2 \\ 0 & \text{else} \end{cases}$$

where, α is a parameter that specifies the full filter length.

The windowed upsample filter is found by multiplying the window with the ideal impulse response. The result is similar to the equation for the un-normalized coefficients, c_{ij},. The ideal impulse response and the truncating window for upsampling are illustrated in Figure F-7.

The complexity of the upsampler can be reduced in a similar manner as the downsample case by combining the filtering and sampling operations. Filtering calculations are performed at only the locations that will actually be sampled.

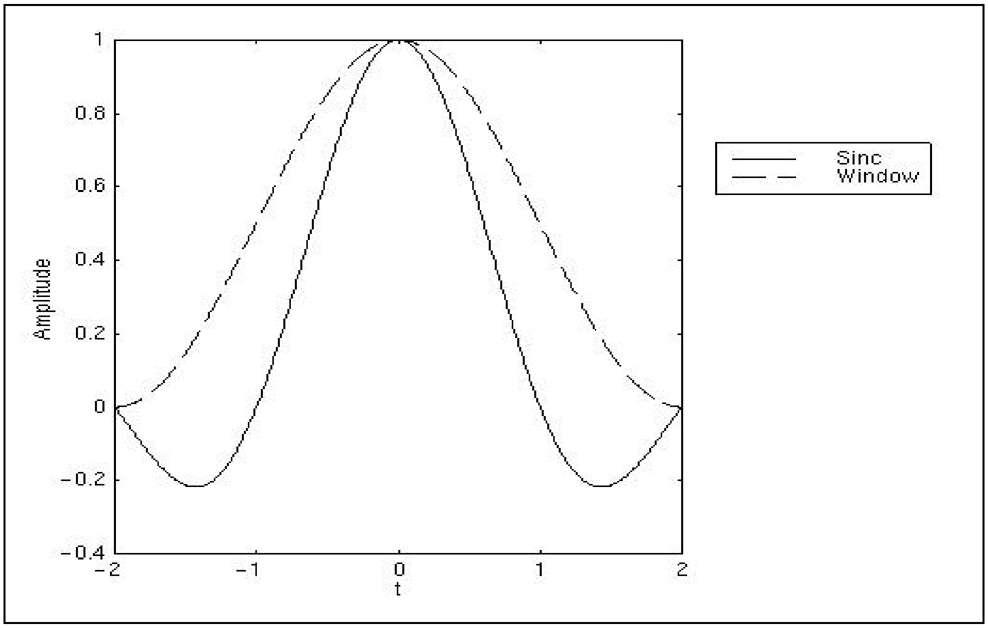

FIGURE F-7 Ideal upsample filter and truncating window.

F.3 Frequency response of practical filters

The following figure shows the superimposed frequency responses of downsample and upsample filter pairs when the filter length parameter, α, is equal to 4, which is the value that is fixed in the normative document. Also shown is the combined response, which is the multiplication of the downsample and upsample filter frequency responses. The downsample and upsample ratios for this example are both equal to 2. The actual filter length for the downsample filter is the length parameter multipled by the downsample ratio; the upsample filter length is the unmodified length parameter. Since the upsample filter, in reality, only influences the frequencies of the original signal in the interval $[0, \pi/R]$, it is displayed in this range so that it can be superimposed with the downsample filter for a meaningful comparison.

The figure shows that the windowed filters provide a sharper cutoff on the downsampling side, and a smoother response (i.e., smaller ripples) on the upsampling side. These characteristics are due to both the differences in filter length and the choice of windows. Placing the filter with a sharper transition on the downsampling side is intuitive since it follows that aliasing will be reduced, and that more of the original signal will be preserved. Furthermore, the design of the downsample filter is more critical because the downsampling operation is the most destructive component of the system. The deliberate choice of windows and filter lengths on both the downsampling and upsampling sides was made in an effort to control the loss of information, and to minimize the computational complexity.

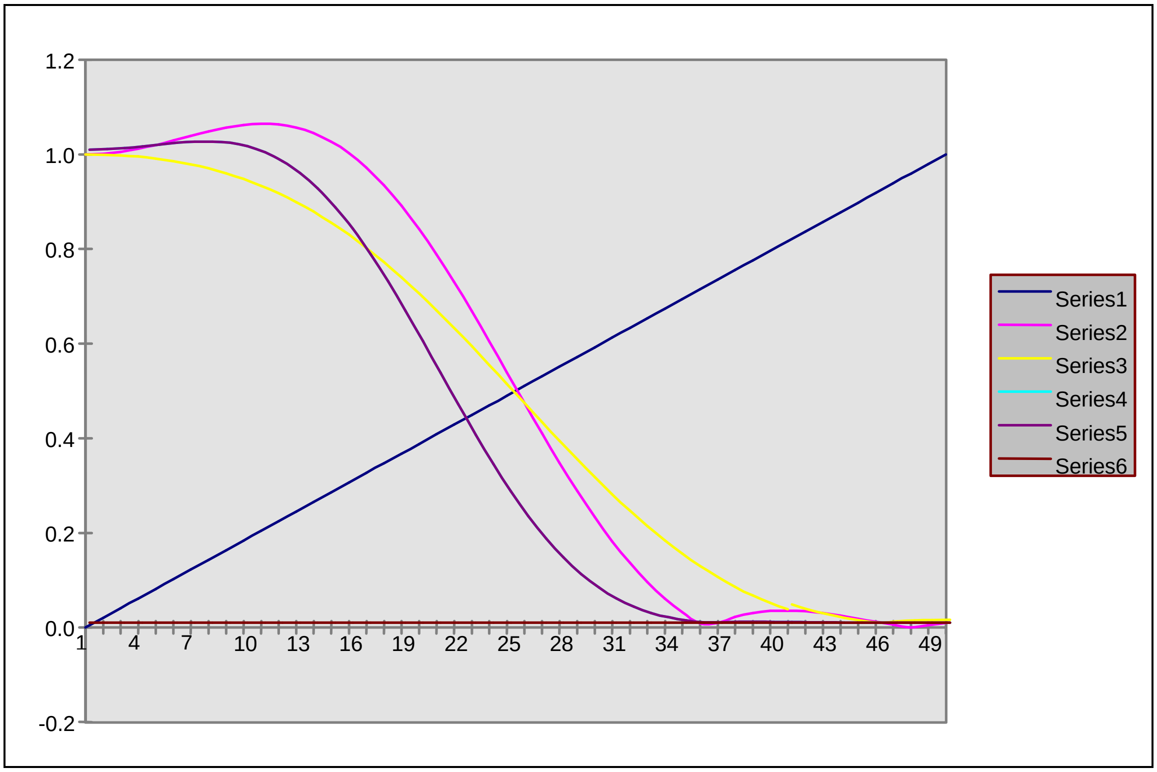

FIGURE F-8 Frequency response of the practical filters using $\alpha = 4$ and $R = 2$.

F.4 Illustration of the filtering operation

A pictorial example of the downsample filtering operation is shown in Figure F-9. The upsample filter can be illustrated similarly with a change in equations and objective. In the example presented, the downsample ratio, R, is 2, the filter length parameter, α, is 4, and the output sample, i, under consideration is 2. The pertinent implementation parameters that need to be calculated for every output sample include the integer indices for the beginning and ending of the finite length filter, and the filter coefficients. Additionally, the filter center, β_i, is automatically calculated when the pertinent parameters are determined. These quantities, with the exception of the coefficients, are labeled on Figure F-9.

The intuitive filter center for the downsampling operation is merely the first term of the equation for β_i: $i \cdot R$. For a downsample ratio that is equal to 2, this would specify that the filter center would be placed, and output samples taken at every other pixel. The rest of the equation represents a shift that is applied to the intuitive filter center. This shift minimizes difficulties at the image edges. For integer downsample ratios, the shift also avoids the situation where a natural zero-crossing of the sinc function falls on a non-zero input data sample. This offset causes the downsampled values to be offset by an amount. The upsampling implementation considers this shift in the calculation of the upsample version of β_i.

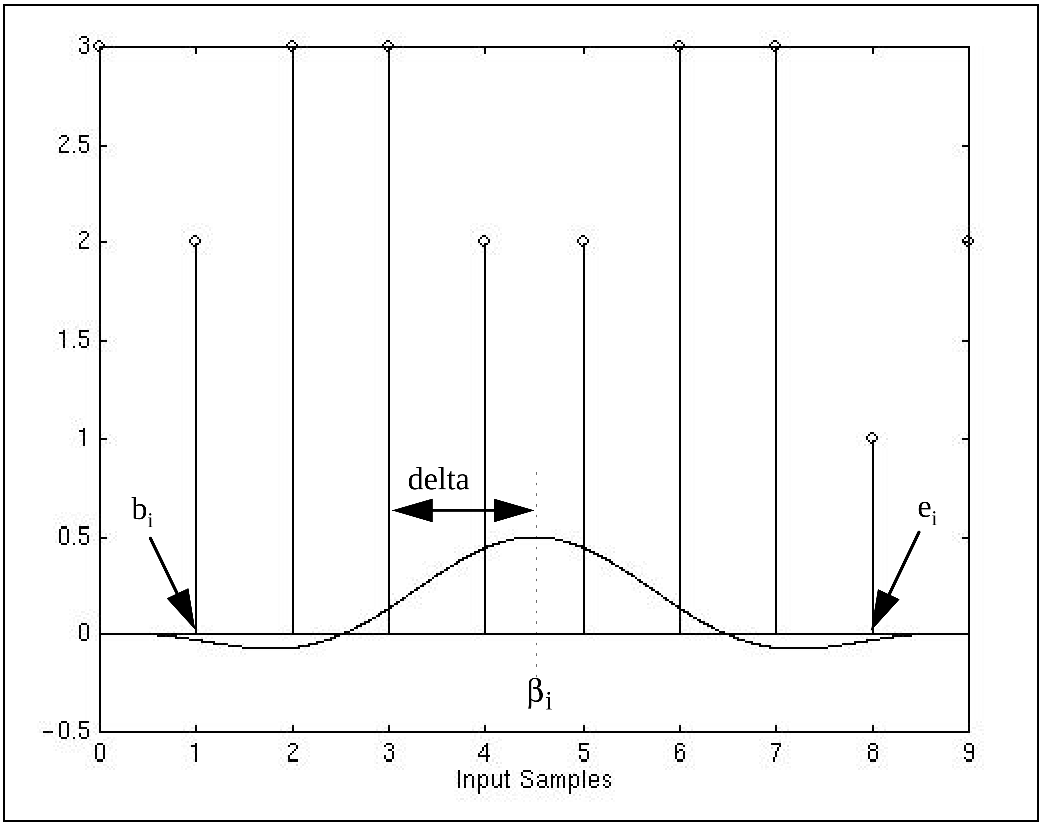

FIGURE F-9 Pictorial view of the filtering operation when α = 4, R = 2, and i = 2

The filters that are implemented are finite in length due to the windows that are applied. The filter beginning and ending points define where the filter is non-zero. These values can be used to determine the number of filter coefficients that need to be calculated, and the input data samples that fall within the filter's non-zero support. The values are integerized since the input data samples fall at integer locations on the original sampling grid.

The filter coefficients are calculated using the equation for c_{ij}. Embedded within this equation is a quantity that determines the distance from a particular input data sample to the filter center. This distance is labeled, delta, in Figure F-9 as an example. The delta quantity is needed in order to calculate the precise value of the downsample filter function for use as a weight on a particular input sample.

Normalization of the filter coefficients, c_{ij}, is necessary in order to avoid the undesirable amplification or attenuation of the output samples. Normalization is performed after all the filter coefficients are calculated for a single output sample. The raw coefficients are summed, and each value is divided by this sum to yield the normalized coefficients, w_{ij}. This normalization procedure assures that the filter response has unity gain at zero frequency, which means that the overall-average image graylevel will not be altered after filtering.